MASTER IN COMPUTER SCIENCE

Defect tracker
a defect tracking CASE tool

Gillmann, Xavier

Award date:
1999

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

Defect Tracker : A Defect Tracking CASE Tool

Xavier Gillmann

Mémoire présenté en vue de l'obtention du grade de

Maître en Informatique

Année Académique 1998-1999

RUE GRANDGAGNAGE, 21 • B-5000 NAMUR (BELGIUM)

Abstract

The aim of Software Engineering is to improve software quality. There are two different ways to consider the quality of a product. The first one is to consider the final product itself by defining a metric and assessing the product according to that metric. The second one is to consider the process that generated the product, expecting that the quality of the final product is somehow influenced by the quality of the design process. Both approaches are complementary and are of interest along this report.

A defect is a flaw or imperfection in a software work product or software process. Defects can affect every component of a software product. Managing them is a part of the design process. The way they are managed in the design process influences the quality of the final product.

This report discusses the design and implementation of a CASE tool designed to manage and track defects. The application, called Defect Tracker, has been designed following the needs of an academic research team. The platform chosen is a Unix system running a web server.

Defect Tracker has an Internet interface and provides facilities to coordinate the defect fixing work. The defect life cycle is internally represented by a customizable set of states. The history of each defect is recorded. When highlighting the same problem, two defect reports can be associated. Defect Tracker implements a hierarchical model of the software product. A responsible group of users is attached to each knot of the hierarchy. Each member of a responsible group recieve an email notification when a defect report is submitted to the unit under the responsibility of the group. An online help is also provided.

Résumé

Le but de l'ingénierie informatique est d'améliorer la qualité des logiciels. Il existe deux manières distinctes d'aborder la qualité d'un produit. La première est de considérer le produit final en définissant une mesure et en évaluant le produit par rapport à cette mesure. La seconde méthode consiste à évaluer le procédé qui a généré le produit, en supposant que la qualité du produit final est influencée par la qualité du procédé de conception. Les deux approches sont complémentaires et d'intérêt tout au long de ce rapport.

Une défectuosité est une erreur ou une imperfection dans un produit logiciel. Les défectuosités peuvent affecter chaque composant d'un logiciel. Gérer les défectuosités fait partie du procédé de conception et la façon dont elles sont gérées influence la qualité du produit final.

Ce rapport traite de la conception et de l'implémentation d'un outil CASE destiné à gérer et tracer les défectuosités. L'application, appelée Defect Tracker, a été conçue en suivant les besoins d'une équipe universitaire. La plate-forme choisie est un système Unix sur lequel tourne un serveur web.

Defect Tracker fournit une interface sur Internet. Il propose également des facilités de coordination de l'équipe de développement face aux rapports d'erreurs. Le cycle de vie des erreurs est modélisé dans l'outil à l'aide d'un ensemble adaptable d'états. L'historique de chaque défectuosité est enregistré. Lorsque deux rapports d'erreurs concernent le même problème, ils peuvent être associés. Defect Tracker contient une représentation interne des projets via une hiérarchie d'unités. A chaque noeud de la hiérarchie est attaché un groupe de personnes responsables. Chaque membre du groupe reçoit une notification par courrier électronique lorsqu'une défectuosité est rapportée à une unité sous sa responsabilité. Une aide en ligne est également fournie aux utilisateurs.

Acknowledgments

This report is the result of a collaboration between the University of Auckland, New Zealand and the University of Namur, Belgium. From the Tamaki Campus, University of Auckland, I would like to thank Dr. S. Manoharan and Dr. Ulrich Günther for their leading advices as well as their welcome. Thank you to Dr. Naji Habra who supervised the work in Belgium. Last but not least, I would like to thank Frédéric Dumont for the sharing of his knowledge of CPAN and CTAN modules.

Contents

List of Figures

Introduction

Software liability problem increases with grows of software controlled products. In order to improve software quality, computer scientists implement supporting applications for the design process. These applications are known as CASE [1] tools. Chapter 1 gives a short introduction to those tools.

We decided to focus on a particular activity of the design process: defect tracking. Chapter 2 details the various aspects of that activity and the possible gain that can be achieved with a supporting tool.

Following this, we present Defect Tracker, the application we designed to support defect tracking. Chapter 3 deals with the requirements: the features we aimed and the boundaries we set ourself.

Chapter 4 summerizes the implementation part of the project. The full documentation is given in appendix.

Defect Tracker is not the only tool supporting defect tracking activity. Chapter 5 presents some other solutions given by other organizations in the area.

Finally our conclusions are given in chapter 6.

[1] CASE: Coputer Aided Software Engineering

Chapter 1

CASE tools

When designing software, computer scientists are going through different activities. Tools supporting these activities are known as CASE [1] tools. One option to support the design process is to build a model of the process and computerize it. Much work has been done in this area. It reveals that there is no unique way to describe a design process [Sommerville, 96]. Waterfall, transformational, prototyping, spiral... every model highlights particular activities, another way to structure them or another point of view from which to consider the design process. Designing tools to support the design process suffers from that diversity.

The alternative approach is to design tools supporting a single activity and provide a way to integrate them afterwards in a development environment. From this point of view, we distinguish two categories of CASE tools: horizontal and vertical ones. The vertical CASE tools support a specific activity of the design process. A compiler, for example, supports programmers by allowing them to express the program in a high-level language instead of writing binary code. Automated interface testing tools are other, more recent, examples. On the other hand, horizontal CASE tools support activities that take place across the whole design process. Version management tools are a good example: they provide functions to maintain all files (and their different versions) of a project. More recently, traceability tools have been designed: they store links between the different components of a software product. When a component has to be modified, the developers can easily spot the other components that could be affected by the modification.

The integration of CASE tools is a major topic for many organizations and software companies (see [Wallnau et al., 91]). The most common description of the integration depicts five classes of integration: platform, presentation, data, control and process integration.

- Platform integration is achieved when the chosen CASE tools are running in the same environment. An environment can either be an operating system or a higher level platform such as the Java Virtual Machine.
- Presentation integration concerns the way objects are presented and handled.

[1] Computer Aided Software Engineering

- Data integration is achieved when CASE tools share the same data or are able to handle data from another CASE tools.
- Control integration concerns the ability of a CASE tool to trigger (or control) actions provided by another CASE tool.
- Finally, process integration concerns the enactment and control of the design process through the different CASE tools.

Chapter 2

Defect Tracking

The purpose of this chapter is to provide a basic understanding of the concepts and activities connected with defect tracking. We therefore start by providing a definition of software quality and software defect in section 2.1.

This is followed by the description of the activities involved in defect tracking. The first one, described in section 2.2, is gathering defect reports.

While fixing the defect by changing the software product is the core activity resulting from a defect report, many time is spent coordinating the work around the defect report. The section 2.3 presents a way to tackle the problem.

In section 2.4, we develop the profit that can be acquired from computer aided defect tracking in terms of software product and software process quality.

2.1 Software Quality and Software Defect

The quality of a software work product is a rather difficult concept to define. In 1978, Boehm et al. [Boehm et al., 78] defined quality via a set of characteristics: understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structuredness and efficiency. With the advent of Object Oriented programming, the accent is set on reusability. Nowadays, due to network growth, security is a critical point. Instead of giving a never exhaustive set of characteristics, we will rather limit ourself to the more generic definition of the International Organization for Standardization [ISO] quoted in [Andersson et al., 92] to be: "The totality of features and characteristics of a software product that bear on its ability to satisfy stated or implied needs".

In [Florac, 92], Florac defines a defect as "any flaw or imperfection in a software work product or software process".

We understand a flaw (or imperfection) as difference between a product and a stated or implied need.

The software design process is the "set of activities, methods, practices and transformations that people use to develop and maintain software work product"[Paulk et al., 91].

A software product is the result of any activity of the software design process. The

executable code is an example, and defects associated with that product are often referred as bugs. But defects also concern the documentation associated with the code, the user manual of the software,...

Note that a defect is not defined as a difference between a product and any reference document. Such a definition presumes that a defect-free reference document exists. We assume that any reference document is a part of the software work product and therefore subject to defects. A defect is thus a difference between a product and what it is expected to be or to do. This definition is far from formal. However, we accept it for its convenience to the needs of this project.

2.2 Defect Gathering

The activities that reveal defects are varied. The use of the software product is probably the most obvious one. The conceptual design, for example, uses the requirements specification as a reference but may highlight contradictions in it. Inspecting and testing are other activities that may reveal defects in a software product. Those activities are spread over the whole design process. As a result, defect tracking tools are horizontal CASE tools.

Gathering defect reports can be done by several ways: from automated gathering methods (where defect reports are generated by other CASE tools) to manual ones (where each defect report is submitted by a somebody).

2.3 Defect Management

Managing defects consists, awongst other things, in coordinating the work resulting from a defect report. The central point of this activity is the defect life cycle and the functions derived from it. The defect life cycle has to be explicit enough to indicate the progress of the defect. But a too detailed life cycle may be too constraining for the developers.

Defects are not always independent of each other. Two defect reports may describe the same problem. Defects are also sometimes linked by a cause-effect or a hierarchical relationship. To represent those relationships in the supporting tool is a matter of ratio usefulness/constraint. Note that as the defect database grows, query and search capabilities become necessary to easily spot those relationships.

In order to coordinate efficiently the work, responsibilities have to be defined. The tool supporting defect tracking may depict those responsibilities and manage an access control to enforce them.

Managing defects leads to changes in the software product. Change and version management are separated activities. However these activities are so strongly correlated that it is sometimes easier to consider them as a whole.

2.4 Reports and Metrics

Defects are easy-to-count entities and, following the attributes recorded with the defects, various reports can be elaborated and metrics calculated. Reports can, for example, depict trends in defect discovery, closure,... From those reports, one try to predict, on statistical grounds, remaining defects. Others [CSST] try to spot the point where developers fixes are creating as many problems as they correct.

Aggregating data from defect reports also delivers information on the design process itself. Defects are indeed the result of a flaw in an activity of the design process. Spotting which activities are the cause of the most expensive (in terms of time, cost or other criteria) defects can help to improve the design process.

Defect measurements are common factors in quality metrics. A metric is "a measure of the extent or degree to which a product possesses and exhibits a certain (quality) characteristic" [Boehm et al., 78]. Many quality characteristics rely on human judgment (e.g. understandability,...). However this is rarely the case with metrics calculated from defect reports. As a consequence, those metrics can easily be automated and become very cost effective. Some organizations (such as IEEE [IEEE]) propose such metrics.

Chapter 3

Defect Tracker Conception

We implemented Defect Tracker as a starting point to investigate the field of defect tracking CASE tools. This chapter explains the choices we made during its conception. The first section presents the context into which Defect Tracker had to fit.

Section 3.2 describes the objectives we set ourself for Defect Tracker.

Section 3.3 lists the information stored in the database of Defect Tracker. It also introduces some terms used in Defect Tracker.

A defect in a software product can be seen from two opposite points of view: the one of the person who encounters it and the one of the person who has to fix it. Defect Tracker supports the basic activities of both actors and establishes a communication scheme between them. Section 3.4 describes the input (or external) part which is the defect gathering methods we implemented.

Section 3.5 describes the processing (or internal) part which is the support to the users who have to fix the defect.

In order to manage the tool and keep the database coherent, Defect Tracker also contains some administration facilities described in section 3.6. All of them are performed by a third class of actor.

An online help is provided with Defect Tracker. However that part of the project is not presented in this report.

3.1 Context

Defect Tracker was implemented in response to a request of one team of the Tamaki Campus of the University of Auckland, New Zealand. Far away from quality assessment and process management, the team was mainly preoccupied with parallel computing. The researchers were working nearly exclusively on a Unix system, using very basic tools: mainly compilers and a version management tool (CVS: Concurrent Versions System of Cyclic Software).

3.2 Objectives

The Auckland University team was especially interested in having better feedback of the problems encountered by the persons using their products. The more and more spread use of Internet forced itself upon any other means of communication. We also decided to provide the basic functions to designers in order to manage and coordinate the work resulting from defect reports.

Taking the software process itself into consideration is beyond the scope of this project. Indeed, it assumes that the CASE tool has an internal representation of it. This means that the process is at least modelized. In other words, the organization must have reached at least level three of the Capability Maturity Model of the Software Engineering Institute [SEI]. This was clearly not the case. However we are convinced that even low-level maturity organizations can profit from a simple defect tracking system. We therefore restrict the definition of a defect (within Defect Tracker) to any flaw or imperfection in a software product.

Defect Tracking tools often provide reporting capabilities and quality metrics based on statistical methods. Defect Tracker does not implement any reporting functions and no quality metric is provided. No integration with other CASE tool is foreseen. Defect Tracker is a stand-alone tool. Except the optional use of email and, for one version, the use of Oracle as DBMS [1].

3.3 Database Content

Defects always concern a particular sofware product. A software product can be made of several other software products. In Defect Tracker, a software product is called a unit. A unit has a name and a release. A unit can be made of several units but contributes to at most one unit. A unit which does not contribute to any other is what we usually call a project.

As previsiously said, there are three classes of users accessing the tool: the ones who submit defect reports (called Users), the ones who manage the defects (called Unit Reponsibles, UR in short) and the ones who manage the tool (called Administrators). Unit Responsibles and Administrators have to be registered in the database to perform their tasks while Users can be anonymous. The profiles recorded in the database contain an identifiant login name and a password. The surname, firstname and email address can also be stored.

In order to respect confidentiality of the developers, the Unit Responsibles are member of groups through which they are contacted. A unit has always one and only one group of responsible persons. A group can be responsible for several units.

The database stores the following information about defects: an identifiant, the title, the state of the defect in its life cycle, the priority level of the defect, the description. Optionally the database stores the activity that discovered the defect (test, normal use,

[1] DBMS: DataBase Management System

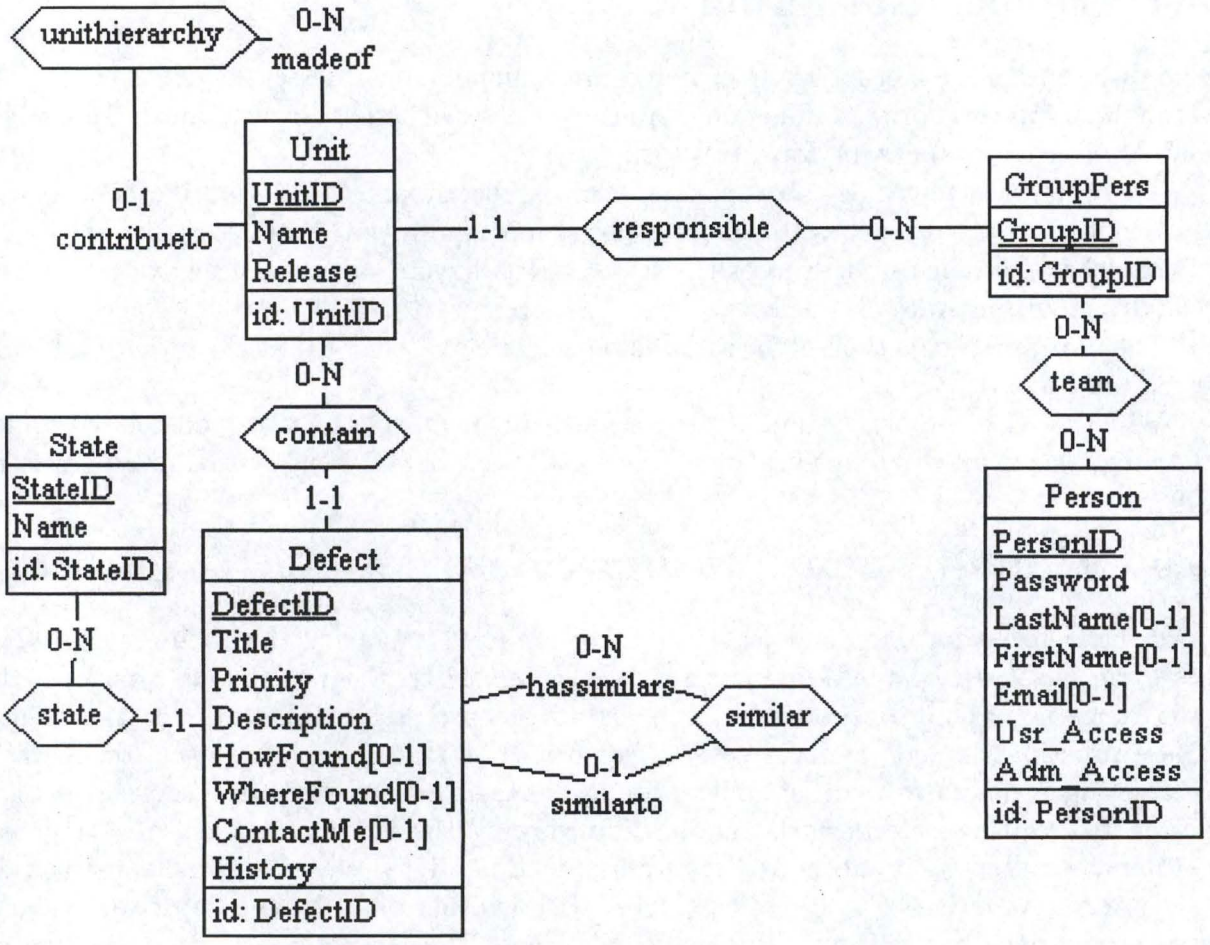


Figure 3.1: Database Conceptual Schema

inspection,...), the environment where the defect was found, and the email address of the person who discovered the defect if that person wants to be contacted during the defect evolution. The database stores also the history of each defect and similar. Every modification to the defect (or similar) is recorded with the time and date it was performed. An optional comment can be added to the description by the person performing the change. A defect always concerns one and only one unit.

Some defect reports concern defects already discovered that are being processed. These defects are called similars. A similar is always associated with only one defect. The values of the state and priority attributes of a similar are the same as the ones of the defect to which they are linked. A similar also always concerns the same unit as its reference defect.

Figure 3.1 summerizes the database structure in an ERA[2] model.

[2]ERA: Entity, Relations and Attributes

3.4 Defect Gathering

The data gathering methods implemented are manual. Each defect report is submitted through an Internet form by somebody. All three classes of actors (Users, Unit Responsibles and Administrators) can perform this task.

There are two methods to submit a defect: a generic and a unit-specific one. In the generic method, the user has to choose a target unit and give the values of the attributes of the defect (detailed 3.3). Except the state attribute which is automatically set to 1 (by default the initial state of a defect).

The unit-specific method is the same as the generic one except that the target is already given (and fixed).

When a defect report is submitted to a unit, the members of the responsible group for the unit recieve an email notification. This feature can be disabled by the administrator.

3.5 Unit Responsible Functions

Tracking defects does not end when they are reported. A defect has a life cycle. The life cycle is set up by the administrator. Only persons in the responsible group for the unit concerned (Unit Responsibles) can access the processing functions for the defect. These functions include transfering a defect from one state of the life cycle to another, retargeting a defect to another unit, viewing and editing the similars associated with a defect, associating defects with another and viewing the history of a defect. A defect can be associated with another defect from any state. This relationship is transitive. In other words, when a defect (A) is associated with another one (B), the defects previously associated with that defect (A) are associated with the new target (B). An associated defect can be disociated at any time, retrieving the inital relationship situation. The state and priority attributes of the dissociated defect do not retrieve their initial values but are set to the value of the corresponding attributes of the previously linked defect.

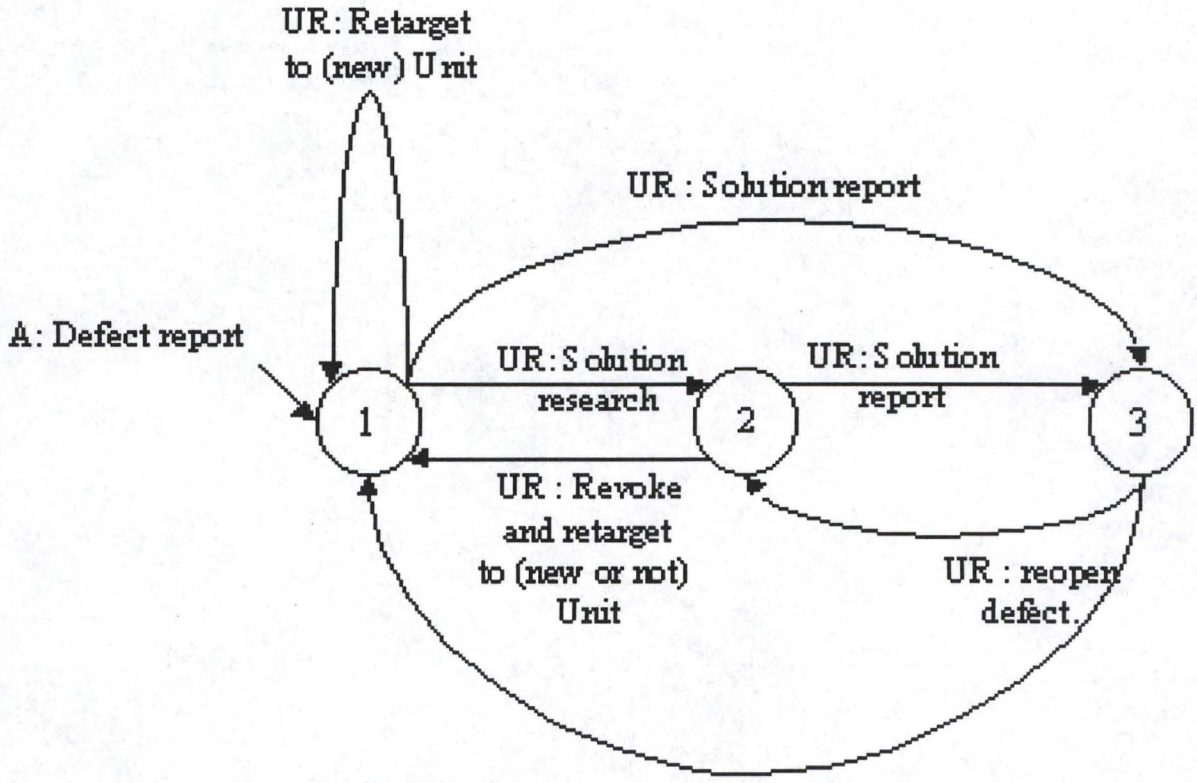
Figure 3.2 describes the defect life cycle we suggest. When created, a defect is active (state 1). When the UR targets the defect to another unit, the defect stays active (state 1). The UR has the opportunity to close the defect (state 3) by providing a solution. He can also assign the defect to himself (state 2). Once the defect is in state 2, the UR can close it (state 3) or set the defect back to the active state (state 1). A closed defect can be reopened and becomes either active (state 1) or assigned (state 2).

Finally the UR has the opportunity to correct (modify) every attribute of the defect. Note: the description of every change (from one state to another or attribute value modification) is recorded with an optional comment.

3.6 Administrator Functions

The administration functions are designed to maintain the database in a coherent state. The user has to be registered as an administrator to be able to access these functions.

They involve: creating, modifying and deleting personal profiles, groups and units. The administrator can also modify the defect life cycle by creating, modifying and deleting states. Finally the administrators can delete defects or similars from the database.
Note: the creation of defects belongs to the input part (detailed in section 3.4). Modifying defects and similars as well as creating similars from defects belongs to the processing part (detailed in section 3.5).



1 : Active state

2 : Assigned state

3 : Closed state

Figure 3.2: Defect Life Cycle

Chapter 4

Defect Tracker Implementation

We presented the conceptual design of Defect Tracker in chapter 3. This chapter summerizes its implementation. The full documentation for the Defect Tracker implementation is given in the appendix. It includes the logical design (A), the physical design (B) and finally the code itself (C). This chapter follows the same structure and summarizes each component.

4.1 Logical Design

This document describes the solution that implements the conceptual design. The description is logical in the sense that it does not take the technical aspect of the solution into consideration. It depicts all the tasks that have to be implemented and how they are linked to each other. For the database, we chose the relational model.

We also designed a library grouping many parameters of the software. The parameters concern several points. The optional use of the email software for defect notification is the object of one parameter. Some parameters concern variables linked to a particular installation of the software: the path to some files (database and scripts)... Finally, some parameters concern layout options: messages displayed, button captions, font color,...

4.2 Physical Design

The physical design takes the technical aspect of the solution into consideration. One of the objectives was to give Defect Tracker an Internet interface. Because of the work environment of the team (see 3.1), we decided to implement CGI [1] scripts on a Unix server. The maintenance and evolution easiness led us to Perl language. We chose to design a Perl script for each task presented in the logical design.

As access control, we decided to use the basic level of security provided by the web server. The only access right not implemented that way concerns the restriction for a Unit

[1] CGI: Common Gateway Interface

Manager to only perform actions on defects submitted to units under his responibility. This access right was implemented within the script that gives access to those functions.

We chose SQL to implement the database. There are two versions of the tool. One (called Defect Tracker XS) does not use any SQL server: we decided to implement a small library to handle flat files database from the SQL commands used in the program. This library does not implement the whole set of SQL statments. The other version (called Defect Tracker OP) uses Oracle as SQL server.

4.3 Code

The code is a list of all source code files written to implement the physical design. The files are split up in three directories related to the three classes of actors: the functions linked to the Users (respectively the Unit Responsibles and Administrators) are grouped in the "pub" (respectively "gware" and "admin") directory. For Defect Tracker XS, we implemented a small SQL library in a Perl package called "XSprite". This package is derived from Shishir Gundavaram's "Srite v3.1" package distributed by CPAN [CPAN]. Most of the library has been changed: some functions have been added, others removed. Even the structure of the package has been somewhat modified. However some parts remain untouched and the XSprite package would not be what it is without the Sprite package as starting point. For Defect Tracker OP, we used Oraperl: a Perl package also distributed by CPAN [CPAN].

It is well known that CGI scripts are a major source of security holes. In order to reduce risks of damages on the server side, we followed the recommendations of W3C [W3C] for CGI and Perl scripting. We also took care about HTML SSI (Server Side Includes) by preventing the end-user to submit HTML tags that would be interpreted later on.

Chapter 5

Existing Tools

Defect Tracker is certainly not the first CASE tool supporting defect tracking. This chapter will give you an idea of the various products that can be found, their particularities and differencies from Defect Tracker.

The products examined can be divided in two categories. The first one, illustrated in sections 5.1 and 5.2, groups tools focusing on team management. The range of these products is often an Intranet. Defect reports are mainly submitted by members of the team (testing team,...). The standard customer of these products are high maturity level organizations. Report and metric capabilities are of outstanding importance.

The second class of tools focuses on the web interfaced defect gathering method. The tools of this category rarely support team coordination. Some of them provide report capabilities. Sections 5.3 and 5.4 give examples of such tools.

5.1 MetaQuest Software Inc.: Census

MetaQuest Software Inc. [MetaQuest] supplies a complete defect tracking CASE tool running on a Windows Intranet. Query and search capabilities are implemented as well as predefined and customizable reports (textual and graphical). Files can be attached to complete the description field of the defect report.

However no Internet interface is provided. The product is only in use within the Intranet of the company. No relationship between defects are foreseen. There is no internal representations of the various branches of a project but the tool supports several project (selected at the login prompt).

5.2 Alexsys Corporation: Alexsys Team 98

Alexsys Team is not a defect tracking tool strictly speaking. Alexsys Corporation [Alexsys] presents the product as "The Team Management System for Windows". All the software is based on the exchange of "work requests" between partners. However, the tool is also presented as a defect tracking tool. Indeed, a work request is just more generic (in the

scope) than a defect report but it is described by the same attributes (status, title, description,...). Alexsys Team 98 offers various report capabilities. Search engine is provided on the database.

The Internet interface is conceived to allow registered users to access the tool easily but no anonymous end-user can submit a work request. No relationship between work requests is implemented.

5.3 Soffront: TrackWeb

Soffront [Soffront] online defect tracking tool is a web interfaced database with query and customizable report capabilities. Files can be attached to defect descriptions. Despite some insignificant ergonomic mistakes, the interface is well designed and easy to handel.

The product supports several projects but does not have an internal representation of it. No team management fonction is provided. The database does not store the whole history of the defects: only the date of the last changed is recorded.

5.4 Hard Boiled: Online Bugbase

Hard Boiled's Online Bugbase [HardBoiled] is a web interfaced database collecting defect reports. The tool offers search and very limited report capabilities. When the tool is not used on the Internet, files can be attached to the defect description.

The database is not multi-project: all defect reports for every project are recorded without distinction. No team management function is implemented. The interface is quite poor: many ergonomic rules are violated and not always relevant information is asked (e.g. screen resolution,...). The history of the defect is not recorded except the date of the last update.

Chapter 6

Conclusions

This report started by a short introduction to CASE tools. After what we outlined the place and the importance of defect tracking within the design process. We presented Defect Tracker: the software we implemented to illustrate the basic functions expected from a defect tracking tool. Following this, we presented some other products and ways to tackel the problem.

To improve the design process is a learning activity. Steps have to be climbed up one at a time. This project shows that defect tracking can be a good starting point for computer aided software engineering. The New Zealander team required a better feedback from their end-users. With Defect Tracker, they also gained a better coordination.

We hope that this report led you to critically examination of the way defects are handeled in your organization. Could this report also bring you some replies on the expectations you can fulfill with defect tracking tools.

Bibliography

[Alexsys] Alexsys Corporation: `http://www.alexcorp.com`

[Andersson et al., 92] Andersson T., Törn A., "Measurement of Software Quality", Abo Akademi University, Department of Computer Science, DataCity, SF-20520 Turku, Finland. March 8, 1992.

[Boehm et al., 78] Boehm, B.W., Brown J.R., Kaspar H., Lipow M., Maclead G.J., Merrit M.J., "Characteristics of Software Quality, Volume 1", TRW Software Series, North-Holland. 1978.

[CPAN] Comprehensive Perl Archive Network:
`http://www.perl.com/CPAN-local/README.html`

[CSST] CSST Technologies: `http://www.csst-technologies.com/`

[Florac, 92] William A. Florac, "Software Quality Measurement : A Framework for Counting Problems and Defects", *Technical Report CMU/SEI-92-TR-22*, September 1992.

[HardBoiled] Hard Boiled: `http://www.hardboiled.com`

[IEEE] IEEE: `http://www.ieee.org/`

[ISO] International Organization for Standardization:
`http://www.iso.ch/index.html`

[MetaQuest] MetaQuest Software Inc. `http://www.metaquest.com`

[Paulk et al., 91] Paulk, Mark C., Curtis, Bill & Chrissis, Mary Beth, "Capability Maturity Model for Software", *Technical Report CMU/SEI-91-TR-24*, 1991.

[SEI] Software Engineering Institute:
`http://www.sei.cmu.edu/`

[Soffront] Soffront: `http://proxy.soffront.com`

[Sommerville, 96] I. Sommerville, "Software Engineering", 5th Edition, Addison-Wesley, 1996.

[Wallnau et al., 91] Kurt C. Wallnau & Peter H. Feiler, "Tool Integration and Environment Architectures", *Technical Report CMU/SEI-91-TR-11*, May 1991.

[W3C] The World Wide Web Security FAQ:
`http://www.w3.org/Security/faq/www-security-faq.html`

Appendix A

Logical Design

Defect Tracker is implemented as an extension of a web server. The client interacts with Defect Tracker by receiving (resp. submitting) HTML forms that are generated (resp. treated) on scripts triggered on the server.

A.1 Database

The database model chosen for the project is the relational model. We derived the relational schema (figure A.1) directly from the conceptual schema (figure 3.1). Except for the "History" attribute of the Defect entity and the "Password" attribute of the Person entity. The defect history is recorded in a log file. The name of the file is the "DefectID" attribute of the defect. All log files are stored in a specified directory of the project. The modification description of any change will be appended to the files. The users passwords are not stored in the database. We decided to use the web server features to manage them.

A.2 Functions

The module hiearchy is presented in figure A.2. The modules are all functional. Data are directly managed from and into the database through the SQL library.

OS: The OS module groups the functionalities accessing the operating system.

Uses: $\emptyset$.

Services: All the usual OS statements. Including reading, writing, creating, deleting and renaming files. As well as date and time functions.

SQL library: The SQL library groups all the access functions to the relational part of the database. As its name indicates it, the interface language chosen is SQL.

Uses: OS.

Services: Connect, Commit, Rollback, Select, Update, Delete and Insert SQL statements.

Access right: The Access right module groups the functionalities used to identifiate the person accessing the database and ensure the access right to the differents part of the database. However no use is made of the database unit. Indeed the password are not stored in the database and are the object of a separated treatment.

Uses: OS.

Services:

Identification: Check the identity of the user.

AccessGWare: Check the access right to the treatment part (unit manager features) of the project.

AccessAdmin: Check the access right to the administration functionalities.

AccessDefect: Check wherever the selected defect is in a unit under the responsibility of the current user.

Person: The Person module groups all the functionalities used to manage the Person table of the database. To access the provided services, the user has to be authentified as an administrator.

Uses: SQL, Access right.

Services:

Create: Create a new record in the Person table and set up the access rights for the profile.

EditAll: Display all the Person entities and provide a selection feature.

Edit: Edit a Person in order to change one of its non-identifiant attribute.

Correct: Change the value of the following fields: FirstName, LastName, Password and Email. The access rights can also be modified.

Delete: Delete a selected Person entity.

State: The State module groups all the functionalities used to manage the State table of the database. To access the provided services, the user has to be authentified as an administrator.

Uses: SQL.

Services:

Create: Create a new record in the State table.

EditAll: Display all the State entities and provide a selection feature.

Edit: Edit a State in order to change one of its non-identifiant attribute.

Correct: Change the value of the Name field.

Delete: Delete a selected State entity.

Group: The Group module groups all the functionalities used to manage the Group table of the database. To access the provided services, the user has to be authentified as an administrator.

Uses: SQL.

Services:

Create: Create a new record in the table.

EditAll: Display all the Group entities and provide a selection feature.

Delete: Delete a selected Group entity.

AddMember: Add a member to the selected Group.

RemoveMember: Remove a member from the selected Group.

Unit: The Unit module groups all the functionalities used to manage the Unit table of the database. To access the provided services, the user has to be authentified as an administrator.

Uses: SQL.

Services:

Create: Create a new record in the Unit table.

EditAll: Display all the Unit entities and provide a selection feature.

Edit: Edit a Unit in order to change one of its non-identifiant attribute.

Correct: Change the value of the Release field.

Delete: Delete a selected Unit entity.

Defect: The Defect module groups all the functionalities used to manage the Defect table of the database.

Uses: SQL, Similar.

Services:

Create: Create a new record in the table.

EditAll: Display all the Defect entities and provide a selection feature.

EditAllForUnit: Display all the Defect entities targetted to a selected unit and provide a selection feature.

Edit: Check the right access of the user to the Defect and edit it with a menu displaying the possible changes that can be performed.

Correct: Change the value of the following fields: Title, Priority, Description, How-Found, WhereFound, ContactMe. A free text can be added to the log entity to comment the change.

Delete: Delete a selected Defect entity.

DeleteAll : Delete all defects and similars from the database.

ChangeState: Change the state of the defect.

Retarget: Change the Target attribute to a new Unit.

Assimilate: Transform the Defect entity to a Similar entity linked to a selected Defect entity.

ViewSimilars: Display all the Similar entities linked to a particular Defect with a selection feature.

ViewHistory: Display the DefectLog entity linked to the selected Defect.

Similar: The Similar module groups all the functionalities used to manage the Similar table of the database.

Uses: SQL, Defect.

Services:

Create: Create a new record in the table.

EditAll: Display all the Similar entities and provide a selection feature.

EditAllForDefect: Display all the Similar entities linked to a specified Defect entity and provide a selection feature.

Edit: Edit the selected Similar entity with a menu displaying the possible actions that can be performed.

DeleteAll: Delete all the Similar entities from the database.

Delete: Delete a selected Similar entity.

Dissociate: Dissociate the Similar entity from its associated Defect. This means deleting the similar and create the corresponding defect. The missing fields are completed by the associated Defect fields values.

ViewHistory: Display the SimilarLog entity associated to the selected similar entity.

Administration: The Administration module is an interface module presenting and coordinating the different functionalities used by the administrators.

Uses: Person, Group, Unit, Defect and Similars.

Service:

DisplayMenu: Display the menu used to access the different functionalities.

Submit Defect: The Submit Defect module display the form used to submit a defect for a selected unit.

Uses: Defect.

Service:

FormForUnit: Display the input form to describe the defect for a selected unit.

GroupWare: The GroupWare module is an interface module presenting and coordinating the functionalities used by the end-users. When calling the Edit service from the Defect module, the module check the access right for that defect.

Uses: Defect, Similars,Unit Navigate and Access right.

Service:

ReportForUnit: Display the defect report for a selected unit and the menu used to access the different functionalities.

Unit Navigate: The Unit Navigate module groups the functionalities used to navigate through the unit hierarchy of the Defect Tracker database.

Uses: SQL, Submit Defect.

Services:

DisplayHierarchy: Display the whole unit hierarchy in the repository of Defect Tracker and provide a way to navigate through it.

SelectUnit: Provide a way to select a unit.

Main menu: The Main menu module display the access to the three groups of functionalities (i.e. Submit a defect, Unit manager and administrator functions) and check the right access for each when one is selected..

Uses: GroupWare, Submit Defect, Administration and Access right.

Services:

DisplayMenu: Display the menu used to enter the requested session.

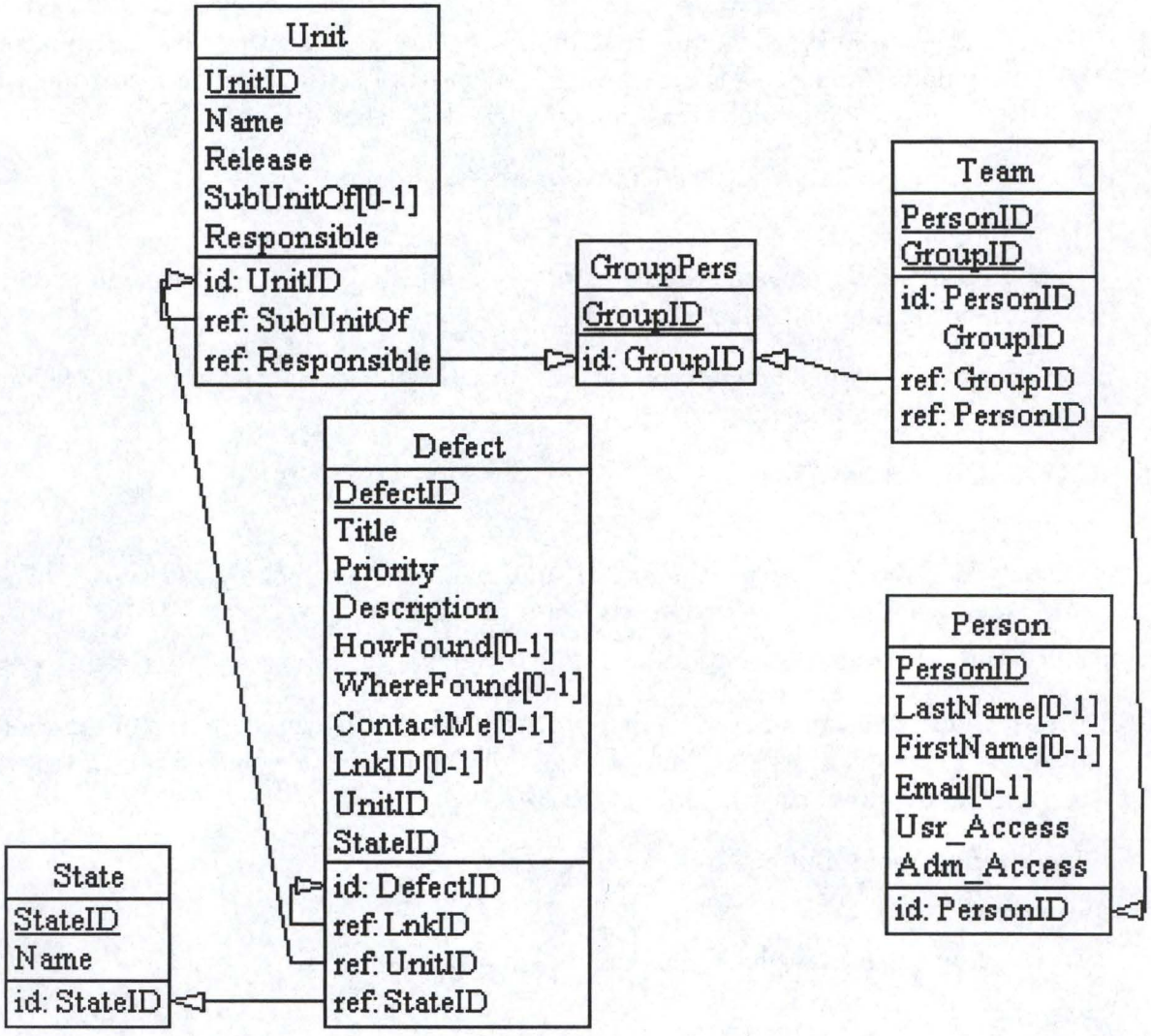


Figure A.1: Repository relational schema

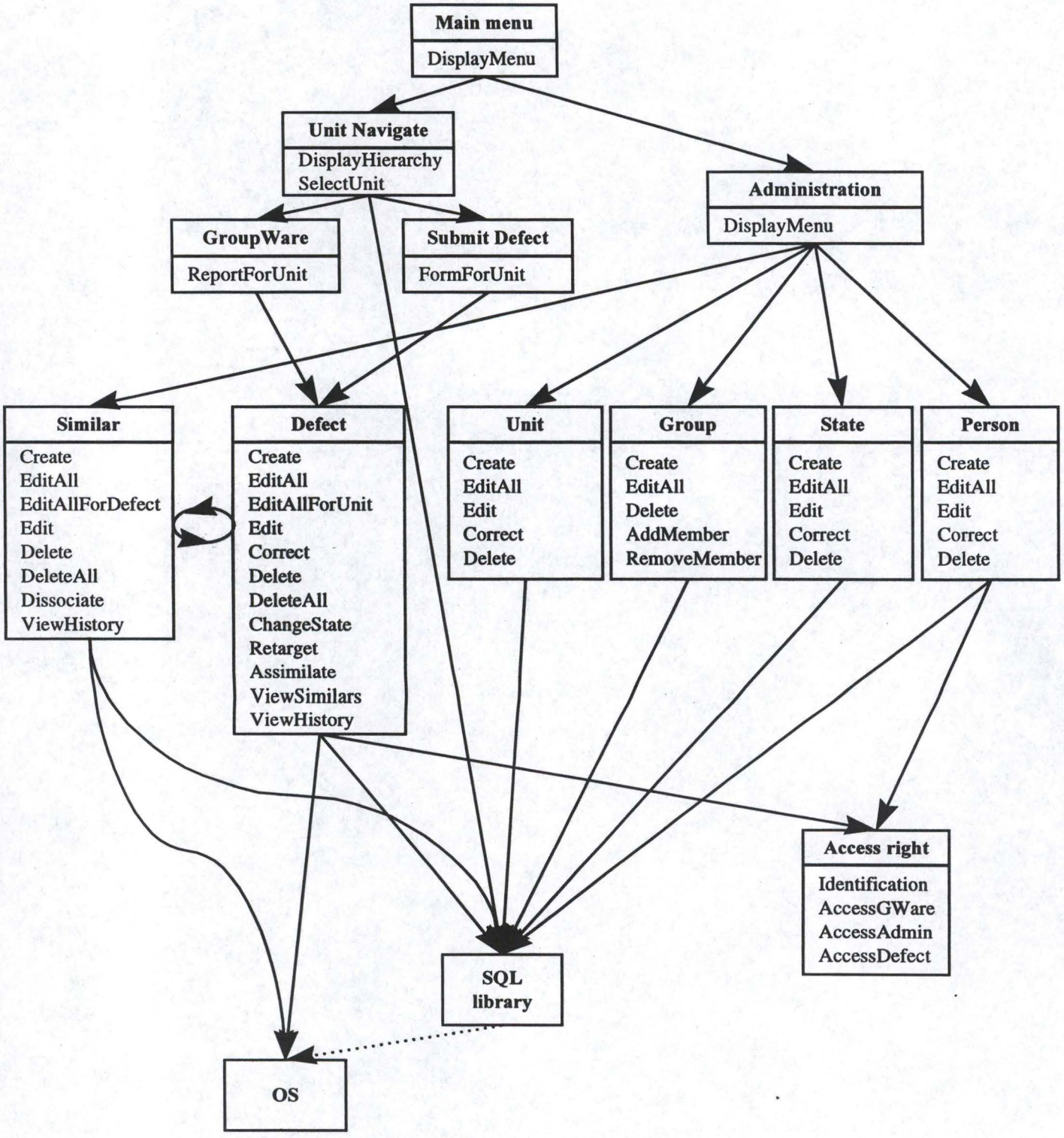


Figure A.2: Logical Architecture

Appendix B

Physical Design

Defect Tracker is implemented with CGI scripts written in Perl language. There are two versions of Defect Tracker. Defect Tracker XS implements a small perl library accessing flat files from SQL commands while Defect Tracker OP uses Oracle as DBMS. This chapter presents Defect Tracker OP. The main differences from Defect Tracker XS are nevertheless presented. The code files are divided in three main directories:

- The **pub/** directory groups all the scripts implementing the data gathering functions as well as libraries used in several other scripts. The section B.1 details this part.
- The **gware/** directory groups all the scripts implementing the Unit Responsible functions. The section B.2 deals with those scripts.
- The **admin/** directory groups all the scripts implementing the Administrator functions. See section B.3 for details.

The gware and admin directories are protected by a ".htaccess" file. The ".htaccess" file of each directory references a common ".htpasswd" and a ".htgrp" in the root directory of Defect Tracker. Indeed no package has been specially designed to check the access rights. The access rights are checked by the web server when the person attempt to access the script that execute the command triggered. Some scripts (PersonCreated.cgi,...) update the ".htpasswd" and ".htgrp" files. The access rights for these files have to be set up properly for the identity under which the CGI scripts are running on the server. The only access right the server could not check is the *AccessDefect* right presented in the Logical Design. However we decided not to create a library for that and implemented the check inside the script that need it (*EditDfct.cgi*).

In order to prevent end-users to use HTML tags (and particulary HTML SSI) in their input, every "<" in the data submitted by users is replaced by "%lt", replacing the HTML tags by their string equivalents in a HTML context.

The code files can be classified in two categories: the packages and the scripts themselves.

- The **packages** are libraries grouping functions used in several scripts. They are recorded in the *lib/* directories of each main directory.

- The **scripts** themselves are procedural: they receive a (CGI) string as input and generate an HTML file sent to the client.

The *db/* directory contains all the database files. The SQL script used to create the database as well as some other usefull SQL scripts (a sample set of data,...). For Defect Tracker XS, the directory contains the database files. A text file is foreseen for each table of the relational model. The data are stored in their CGI form. The field separator is "::". Following CGI standard, the ":" character is coded by its hexadecimal code. This feature ensures that no confusion is made between the data and the separator. For both versions of Defect Tracker, the *db* directory also contains the log files for each defect and similar in the *dfcthist/* directory. As the ".htaccess" files, this directory has to be writable by the scripts.

Finaly the *gifs/* directory contains the various images displayed in the HTML file as well as the welcome page of Defect Tracker.

B.1 "pub" Directory

lib/DisplaySet.pm: The package is used to display, in a HTML context, the set of data returned by a SQL command given to the Oraperl library.

lib/Email.pm: This package centralizes the email warning method in order to easily keep control of this feature. Indeed sending mails from scripts is a well known source of security problem.

lib/Env.pm: This package centralizes the needed parameters to access Oracle from the CGI scripts.

lib/Param.pm: In order to make *Defect Tracker* as customisable as possible, many parameters are initialised into variables by this library. The variables concern, the optional mail warning feature, the physical files names and path for the database and some layout options.

lib/UnitHierarchy.pm: The navigation through the unit hierarchy is used in several places in the software: to select the target when submiting a defect, to consult the defect reports of a particular unit and to assign a desfect to a new target. This package provides a function displaying the hierarchy and providing navigation and selection features. The HTML code generated contains a Java script using some DHTML characteristics provided by Internet Explorer 4 but not supported by Netscape 4.6.

DefectCreated.cgi: This script implements the *Create* function of the *Defect* module presented in the logical architecture.

ReportDefect.cgi: This script generates the first window of the generic data gathering method. This includes the unit hierarchy.

ReportToUnit.cgi: This script generates the HTML form used to submit a defect to a selected unit.

UnitHierarchy.cgi: This script displays the unit hierarchy, calling the "UnitHierarchy.pm" library.

B.2 "gware" Directory

lib/ReportForUnit.pm: The ReportForUnit package displays the defect reports for a selected unit.

ChangeStt.cgi: This script implements the *Change State* functionality of the *Defect* module presented in the logical architecture.

Correct.cgi: This script implements the *Correct* functionality of the *Defect* module presented in the logical architecture.

DefectReport.cgi: This script displays the first window of the Unit Responsible part of Defect Tracker. This includes the unit hierarchy.

EditDfct.cgi: This script implements the *Edit* functionality of the *Defect* module presented in the logical architecture.

EditLink.cgi: This script implements the *Edit* functionality of the *Similar* module presented in the logical architecture.

LinkHist.cgi: This script implements the *View History* functionality of the *Similar* module presented in the logical architecture.

LinkTo.cgi: This script implements the *Assimilate* functionality of the *Defect* module presented in the logical architecture.

ReplyDfct.cgi: This script is used to display the forms where the user can enter the parameters linked to the action they want to performe on a defect (edited by the *EditDfct* script).

ReplyLnk.cgi: This script is used to performe the *Dissociate* or the *View History* actions on a *Similar* entity (edited by the *EditLink* script).

ReportForUnit.cgi: This script displays the defect report for a selected unit.

Retarget.cgi: This script implements the *Retarget* functionality of the *Defect* module presented in the logical architecture.

B.3 "admin" Directory

lib/AdminForm.pm: This package implements the *Display Menu* function of the Administration module of the logical architecture.

lib/EditGrp.pm: This package the displays the form used to change the composition of a group.

Admin.cgi: This script executes the selected operation in the administration menu. It covers the *EditAll* functions of the *Similar*, *Defect*, *Unit*, *Group* and *Person* modules presented in the logical architecture. The script is also executed without any selected action to display the initial administration menu (using "AdminForm.pm" library).

GroupCreated.cgi: This script implements the *Create* functionality of the *Group* module presented in the logical architecture.

PersonCreated.cgi: This script implements the *Create* functionality of the *Person* module presented in the logical architecture.

ReplyGrp.cgi: This script implements the *AddMember*, *RemoveMember*, *Delete* and *ResetAll* functions of the *Group* module presented in the logical architecture.

ReplyPrs.cgi: This script implements the *Correct* and *Delete* functions of the *Person* module presented in the logical architecture.

ReplyStt.cgi: This script is used performe the *Correct* and the *Delete* functions of the *State* module presented in the logical architecture.

ReplyUnt.cgi: This script implements the *Correct* and *Delete* functions of the *Person* module presented in the logical architecture.

StateCreated.cgi: This script implements the *Create* functionality of the *State* module presented in the logical architecture.

TblDfct.cgi: This script implements the *Delete* and *DeleteAll* functions of the *Defect* module presented in the logical architecture.

TblGrp.cgi: This script implements the *Edit* functionality of the *Group* module presented in the logical architecture.

TblLnk.cgi: This script implements the *Delete* and *DeleteAll* functions of the *Similar* module presented in the logical architecture.

TblPrs.cgi: This script implements the *Edit* functionality of the *Person* module presented in the logical architecture.

TblStt.cgi: This script implements the *Edit* functionality of the *State* module presented in the logical architecture.

TblUnt.cgi: This script implements the *Edit* functionality of the *Unit* module presented in the logical architecture.

UnitCreated.cgi: This script implements the *Create* functionality of the *Unit* module presented in the logical architecture.

Appendix C

Code

C.1 "pub" Directory

DisplaySet.pm

```
package DisplaySet;

require 5.000;

use Env;
use Oraperl;
use File::Basename;

sub display {
   my ($csr,$col,$reply,$btn_1,$btn_2,$param,) =@_;                    10
   print "<TABLE BORDER CELLSPACING=0 CELLPADDING=5><TR>";
   foreach (@{$col}){
      print "<TH>$_</TH>";
   }
   $count=0;
   print "<FORM METHOD=\"POST\" ACTION=\"$reply\">" if ($reply);
   if ($param) {
      $param=~tr/+/ /;
      $param=~s/%(..)/pack("c",hex($1))/ge;
      print "<INPUT TYPE=hidden NAME=param VALUE= \"$param\">";          20
   }
   while (($idcol,@data)=&ora_fetch($csr)) {
      $count++;
      print "<TR>";
      if ($reply) {
         print "<TD><INPUT TYPE=submit NAME=line VALUE=$idcol></TD>"
      }
      else {
         print "<TD>$idcol</TD>"
      }                                                                30
      foreach (@data){
```

```
        tr/+/ /;
        s/%(..)/pack("c",hex($1))/ge;
        $_ = "~" if not($_);
        print "<TD>".$_."</TD>";
      }
    }
    print "</TABLE>";
    print "<INPUT TYPE=submit NAME=action VALUE=$btn_1>" if ($btn_1);
    print "<INPUT TYPE=submit NAME=action VALUE=$btn_2>" if ($btn_2);
    print "</FORM>" if ($repℓy);
    return ($count);
}
```

Email.pm

```
package Email;

require 5.000;

use Env;
use Oraperl;
use Shell;

sub email {
    my ($ℓda,$unit,$dfct) =@_;

    $csr=&ora_open($ℓda,"select Responsible from Unit where "
                 ."(UnitID = \'$unit\')");
    ($grpname) =&ora_fetch($csr);
    $csr=&ora_open($ℓda,"select PersonID from Team where "
                 ."(GroupID = \'$grpname\')");
    @data=&ora_fetch($csr);
    foreach (@data) {
      s/\s*//;
      $csr=&ora_open($ℓda,"select Email from Person where (PersonID=\'$_\')");
      @carnet=&ora_fetch($csr);
      foreach $add(@carnet){
        if ($add) {
          $add=~tr/+/ /;
          $add=~s/%(..)/pack("c",hex($1))/ge;

##########
#check the email address
##########
          $add=~ /([\w\-\+]+)@([\w\-\+\.]+)/ || return (0);
          open (MAIL, "| $Param::sendmail");
          print MAIL "To : $add\n";
          print MAIL "Subject : New defect in unit $unit\n\n";
          print MAIL "The unit \"$unit\" received the defect \"$dfct\"!\n";
          close (MAIL);
```

```
        }
      }
    }
}
```

Env.pm

```
BEGIN {
        unshift (@INC, "/home/rouge/students/xgi/lib/perl5/site_perl/sun4-solaris");
        unshift (@INC, "/home/rouge/students/xgi/lib/perl5/site_perl");
        }

$ENV{"ORACLE_BASE"}="/opt/blanc/oracle";
$ENV{"ORACLE_HOME"}="/opt/blanc/oracle/product.13-7/8.0.4";
$ENV{"ORA_NLS"}= $ENV{"ORACLE_HOME"}."/ocommon/nls/admin/data";
$ENV{"TMPDIR"}="/home/rouge/projects/oracle/tmp";

1;
```

Param.pm

```
package Param;

#optional feature
$Param::UserMailWarnin = 0;
# 0 == no e-mail warning
# 1 == mail warning when a defect is created/retargeted

$Param::sendmail = "/usr/lib/sendmail -t -oi";

#path
$Param::RDTPath ="/~xgillman/cgi-bin/defecttracker/";
$Param::DTPath = "/home/rouge/students/xgi/http-pub/cgi-bin/defecttracker/";
$Param::DBPath = $Param::DTPath."db/";
$Param::DfctHistPath = $Param::DTPath."db/dfcthist/";
$Param::PubPath = $Param::DTPath."pub/";
$Param::RPubPath = $Param::RDTPath."pub/";
$Param::AdminPath = $Param::DTPath."admin/";
$Param::RAdminPath = $Param::RDTPath."admin/";
$Param::GWarePath = $Param::DTPath."gware/";
$Param::RGWarePath = $Param::RDTPath."gware/";
$Param::RGifPath = $Param::RDTPath."gifs/";

$Param::banner = $Param::RGifPath."banner.gif";
#layout options
$Param::Thanks = "New defect commited,<BR> thank you for your collaboration";
$Param::btn_OpenDfctReport = "Submit a Defect";
$Param::btn_OpenGroupWareSession ="Open a Unit Manager Session";
$Param::btn_OpenAdminSession = "Open a Administration Session";
```

```
$Param::btn_AdminPerson = "Person";
$Param::btn_AdminGroup = "Group";
$Param::btn_AdminTeam = "Team";
$Param::btn_AdminUnit = "Unit";
$Param::btn_AdminDfct = "Edit Defects";
$Param::btn_AdminLnk = "Edit Similars";
$Param::btn_AdminState = "Defect Life Cycle";

$Param::btn_ChangeState = "Change State";
$Param::btn_ROpen = "ReOpen";
$Param::btn_Sol = "Close";
$Param::btn_Assign = "Assign";
$Param::btn_GWareModify = "Modify";
$Param::btn_GWareLnk = "Same As";
$Param::btn_GWareULnk = "Dissociate";
$Param::btn_GWareVLnk = "View Similar(s)";
$Param::btn_GWareHist = "View History";
$Param::btn_GWareRetarg = "Retarget";

$Param::btn_Create = "Create";
$Param::btn_Reset = "Reset";

$Param::btn_NewTarg = "New Target";
$Param::btn_NewState = "New State";
$Param::btn_Cor = "Submit Change";

#html layout options
$Param::bgcolor = "BGCOLOR=#FFFFFF";
$Param::text = "TEXT=#000000";
$Param::PromptColor = "#00A5A5";
```

UnitHierarchy.pm

```
package UnitHierarchy;

require 5.000;

use Env;
use Oraperl;
use Param;

sub subunitof {
   local ($unit,$href,$item)=@_;
   foreach (@{$UnitHierarchy::point}){
      if ($$_{SubUnitOf} eq $unit) {
         $dUnitID=$$_{UnitID};
         $dUnitID=~tr/+/ /;
         $dUnitID=~s/%(..)/pack("c",hex($1))/ge;
         $dUnitName=$$_{Name};
```

```
        $dUnitName=~tr/+/ /;
        $dUnitName=~s/%(..)/pack("c",hex($1))/ge;

        print "<LI ID=\"fold\">$dUnitName (ID : $dUnitID)</LI>\n";
        print "<UL STYLE=\"margin-left:5mm;display:none\">";
        print "<LI ID=\"item\"><A TARGET=\"contenu\" HREF=\"$href";
        print "UnitName=$$_{Name}&UnitID=$$_{UnitID}\">$item</A></LI>";
        &subunitof($$_{UnitID},$href,$item);
        print "</UL>";
    }
  }
}


sub display {
  ($href,$item,$itemgif) = @_;

print <<END;
<HTML>
<HEAD>
<STYLE>
<!--
#fold{font-size: 10pt;cursor:hand;list-style-image:url(../gifs/fold.gif)}
#item{font-size: 10pt;cursor:hand;list-style-image:url(../gifs/$itemgif)}
//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--

function change(){
  if(!document.all)
    return
  if (event.srcElement.id=="fold") {
    var srcIndex = event.srcElement.sourceIndex
    var nested = document.all[srcIndex+1]
    if (nested.style.display=="none") {
      nested.style.display=''
      event.srcElement.style.listStyleImage="url(../gifs/open.gif)"
    }
    else {
      nested.style.display="none"
      event.srcElement.style.listStyleImage="url(../gifs/fold.gif)"
    }
  }
}
document.onclick=change

//-->
</SCRIPT>
</HEAD>
<BODY>
```

```
END

$ℓda=&ora_login('sid1','xgillman','xgillman');
$csr=&ora_open($ℓda,'select UnitID, Name, SubUnitOf  from Unit');
@result=();
while (($coℓ1,$coℓ2,$coℓ3)=&ora_fetch($csr)) {
        undef $hash;
        $$hash{UnitID}=$coℓ1;
        $$hash{Name}=$coℓ2;
        $$hash{SubUnitOf}=$coℓ3;
        push @result, $hash;
        };
&ora_close($csr);
&ora_logoff($ℓda);
$UnitHierarchy::point=\@result;
print "<UL STYLE=\"margin-left:5mm\">";
$hierarchy=&subunitof("",$href,$item);
print "</UL>";
print "</BODY>";
}
```

DefectCreated.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use Email;
use Time::Local;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        $fieℓds{$key}=~s/%3C/&lt/g;
        }

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
<FONT COLOR=$Param::PromptColor><B>$Param::Thanks</B></FONT><BR>
END

$ℓda=&ora_login('sid1','xgillman','xgillman');
```

```
$csr=&ora_open($ℓda,"select DefectID from Defect");

##########                                                              30
#Compute the identifiant for the new defect.
##########
$max=0;
while (($id)=&ora_fetch($csr)) {
    $id=~s/^dfct//;
    $id=~s/\s*$//;
    $max=$id if  ($id > $max);
   }
$max++;
$dfctid="dfct$max";                                                     40

##########
#Insert the record in the table.
##########
$csr=&ora_open($ℓda,"insert into Defect (DefectID, Title, StateID, Priority, "
            ."Description,UnitId, ContactMe, HowFound, WhereFound,LnkID ) "
            ."values (\'$dfctid\',\'$fields{title}\',\'$fields{state}\',"
            ."\'$fields{prior}\',\'$fields{descr}\',\'$fields{targ}\',"
            ."\'$fields{Cnt}\',\'$fields{HFnd}\',\'$fields{WFnd}\',NULL)");
                                                                        50

##########
#Send an e-mail notification to responsibles.
##########
if ($Param::UserMailWarnin) { &Email::email($ℓda,$fieℓds{targ},$dfctid)}
&ora_logoff($ℓda);

##########
#Create the log file for the defect.
##########
$fiℓe = $Param::DfctHistPath.$dfctid;                                   60
open (HIST, ">$file");
$time=localtime();
print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect <U>created</U> on $time"
        ."</FONT>\n";
close HIST;

print "</CENTER></BODY></HTML>";
```

ReportDefect.cgi

```
#!/usr/local/bin/perl
BEGIN {
       unshift (@INC, "lib/");
       }

use Param;
use UnitHierarchy;
```

```
$src=$Param::RPubPath."UnitHierarchy.cgi?href=".$Param::RPubPath;
$src=$src."ReportToUnit.cgi?&item=New+Form&itemgif=form.gif";                    10

print <<END;
Content-type: text/html\n\n
<FRAMESET COLS="250,*">
  <FRAME MARGINWIDTH="0" MARGINHEIGHT="0" NAME="index" SRC="$src">
  <FRAME MARGINWIDTH="0" MARGINHEIGHT="0" NAME="contenu" SRC="contenu.html">
</FRAMESET>
END
```

UnitHierarchy.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        }

use UnitHierarchy;

$temp = $ENV{'QUERY_STRING'};
@pairs=split(/&/,$temp);
foreach $item(@pairs){                                                          10
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        $fieℓds{$key}=~tr/+/ /;
        $fieℓds{$key}=~s/%(..)/pack("c",hex($1))/ge;
        }

print "Content-type: text/html\n\n";
&UnitHierarchy::display($fieℓds{href},$fieℓds{item},$fieℓds{itemgif});
```

C.2 "gware" directory

ReportForUnit.pm

```
package ReportForUnit;

require 5.000;

BEGIN {
        unshift (@INC, ". ./pub/lib/");
        }
use Env;
use Oraperl;
use Param;
use DisplaySet;

sub display {
    ($UnitID,$UnitName) = @_;

    print <<END;
<CENTER>
<FONT COLOR=$Param::PromptColor><B>Defect Report of  unit </FONT>
$UnitName</B> (ID: $UnitID)<BR><BR>
END

##########
#Load the defect life cycle
##########
   $lda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($lda,'select StateID, Name  from State');
   @lifecycle = ();
   while (($state,$name)=&ora_fetch($csr)){
      $hash ={};
      $$hash{state} = $state;
      $$hash{name} = $name;
      push @lifecycle, $hash;
   }
   @col=(DefectID,Title,Priority);

##########
#Display the summery of the defect for each state of the defect life cycle
##########
   foreach $stt(@lifecycle) {
      $csr=&ora_open($lda,"select DefectID,Title,Priority from Defect "
                  ."where(( UnitID = \'$UnitID\') and "
                  ."(StateID = \'$$stt{state}\')and(LnkID is NULL))");
      $count=0;
      while (@data=&ora_fetch($csr)){
         $count++;
      }
      if ($count==0) {
```

```
        print "<FONT COLOR=$Param::PromptColor><B>No Defect in "
            ."<FONT COLOR=#000000>$$stt{name}</FONT>state.</B></FONT><BR>";
    }
    else {
        $csr=&ora_open($ℓda,"select DefectID,Title,Priority "
                    ."from Defect where(( UnitID = \'$UnitID\') and "
                    ."(StateID = \'$$stt{state}\')and(LnkID is NULL))");
        print "<FONT COLOR=$Param::PromptColor><B>Defects in "
            ."<FONT COLOR=#000000>$$stt{name}</FONT>state.</B></FONT><BR>";
        &DisplaySet::display($csr,\@col,$Param::RGWarePath."EditDfct.cgi");
    }
  }
  &ora_close($csr);
  &ora_logoff($ℓda);
  print "</CENTER></BODY></HTML>";
}
```

ChangeStt.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, ". ./pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
use Param;
use DisplaySet;
use ReportForUnit;
use Time::Local;
use Shell;

$temp=$ENV{'QUERY_STRING'};
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
  ($key,$content)=split(/=/,$item,3);
  $fieℓds{$key}=$content;
  $fieℓds{$key}=~s/%3C/&lt/g;
}

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########
#Update the database table
##########
$ℓda=&ora_login('sid1','xgillman','xgillman');
```

```
$csr=&ora_open($ℓda,"update Defect set StateID = \'$fields{newstate}\' "
            ."where (DefectID = \'$fields{dfctid}\')");
&ora_close($csr);
&ora_logoff($ℓda);

##########
#Update the log file of the defect
##########
$fiℓe = $Param::DfctHistPath.$fieℓds{dfctid};
open (HIST, ">>$file");
$time = localtime();
$tmp = $fieℓds{newstate};
$tmp=~tr/+/ /;
$tmp=~s/%(..)/pack("c",hex($1))/ge;
print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect put to <U>state $tmp</U> "
        ."on <B>$time</B></FONT>\n";
print HIST "<B>Comment :</B>\n<I>".$fieℓds{comment}."</I>\n"
         if ($fieℓds{comment});
close HIST;

##########
#Display the report from the initial unit
##########
$fieℓds{org}=~tr/+/ /;
$fieℓds{org}=~s/%(..)/pack("c",hex($1))/ge;
$fieℓds{name}=~tr/+/ /;
$fieℓds{name}=~s/%(..)/pack("c",hex($1))/ge;

&ReportForUnit::display($fieℓds{targ},$fieℓds{name});
print "</BODY></HTML>";
```

Correct.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, "../pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
use Param;
use DisplaySet;
use UnitHierarchy;
use ReportForUnit;
use Time::Local;
use Shell;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
```

```
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;                                          20
        $fieℓds{$key}=~s/%3C/&lt/g;
        }

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########
#Load the old values of the attributes from the database                  30
##########
$ℓda=&ora_login('sid1','xgillman','xgillman');
$csr=&ora_open($ℓda,"select Title,Priority,Description,HowFound,WhereFound,"
            ."ContactMe from Defect where(DefectID =\'$fields{dfctid}\')");
@data=&ora_fetch($csr);
foreach (@data) { s/\s*$//;}
($oℓdtitℓe,$oℓdprior,$oℓddescr,$oℓdhfnd,$oℓdwfnd,$oℓdcnt)=@data;

##########
#Update the table of the database                                         40
##########
$csr=&ora_open($ℓda,"update Defect set Title=\'$fields{title}\',Description="
            ."\'$fields{descr}\', Priority = \'$fields{prior}\',HowFound="
            ."\'$fields{hfnd}\',WhereFound=\'$fields{wfnd}\',ContactMe="
            ."\'$fields{cnt}\' where (DefectID = \'$fields{dfctid}\')");

&ora_close($csr);
&ora_logoff($ℓda);

##########                                                                 50
#Update the log file of the defect
##########
$fiℓe = $Param::DfctHistPath.$fieℓds{dfctid};
open (HIST, ">>$file");
$time=localtime();
print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect <U>modified</U> on "
        ."$time</font>\n";
print HIST "<B>Modification(s) :</B>\n";
print HIST "<U>Priority :</U> <B>from</B> $oldprior <B>to</B> "
        ."$fields{prior}\n" if ($oℓdprior ne $fieℓds{prior});            60
print HIST "<U>Title :</U> <B>from</B> $oldtitle <B>to</B> "
        ."$fields{title}\n" if ($oℓdtitℓe ne $fieℓds{title});
print HIST "<U>Description :</U>\n<B>from :</B>\n$olddescr\n<B>to :</B>"
        ."\n$fields{descr}\n" if ($oℓddescr ne $fieℓds{descr});
print HIST "<U>HowFound :</U> <B>from</B> $oldhfnd <B>to</B> "
        ."$fields{hfnd}\n" if (($oℓdhfnd ne $fieℓds{hfnd}) && $oℓdhfnd);
print HIST "<U>HowFound :</U> <B>from</B> <I>nothing</I> <B>to</B> "
        ."$fields{hfnd}\n" if (($oℓdhfnd ne $fieℓds{hfnd})&& !($oℓdhfnd));
```

```
print HIST "<U>HowFound :</U> <B>from</B> $oldwfnd <B>to</B> "
        ."$fields{wfnd}\n" if (($oldwfnd ne $fields{wfnd}) && $oldwfnd);
print HIST "<U>HowFound :</U> <B>from</B> <I>nothing</I> <B>to</B> "
        ."$fields{wfnd}\n" if (($oldwfnd ne $fields{wfnd})&& !($oldwfnd));
print HIST "<U>Contact me :</U> <B>from</B> $oldcnt <B>to</B> "
        ."$fields{cnt}\n" if (($oldcnt ne $fields{cnt}) && $oldcnt);
print HIST "<U>HowFound :</U> <B>from</B> <I>nothing</I> <B>to</B> "
        ."$fields{cnt}\n" if (($oldcnt ne $fields{cnt})&& !($oldcnt));
print HIST "<B>Comment :</B>\n<I>".$fields{comment}."</I>\n"
        if ($fields{comment});
close HIST;

##########
#Display the report from the initial unit
##########
$fields{targ}=~tr/+/ /;
$fields{targ}=~s/%(..)/pack("c",hex($1))/ge;
$fields{targname}=~tr/+/ /;
$fields{targname}=~s/%(..)/pack("c",hex($1))/ge;
&ReportForUnit::display($fields{targ},$fields{targname});
print "</BODY></HTML>";
```

DefectReport.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        }

use Param;

$src=$Param::RPubPath."UnitHierarchy.cgi?href=".$Param::RGWarePath;
$src=$src."ReportForUnit.cgi?&item=Report&itemgif=report.gif";
print <<END;
Content-type: text/html\n\n
<FRAMESET COLS="250,*">
  <FRAME MARGINWIDTH="0" MARGINHEIGHT="0" NAME="index" SRC="$src">
  <FRAME MARGINWIDTH="0" MARGINHEIGHT="0" NAME="contenu" SRC="contenu.html">
</FRAMESET>
END
```

EditDfct.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }
```

```
use Env;
use Oraperl;
use Param;
use DisplaySet;                                                    10

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs){
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
}

print <<END;
Content-type: text/html\n\n                                        20
<HTML><BODY $Param::bgcolor $Param::text>
END

$ℓda=&ora_login('sid1','xgillman','xgillman');

##########
#Load defect attributes from the database
##########
$csr=&ora_open($ℓda,"select DefectID, Title,Priority, Description, HowFound,"
            ." WhereFound,ContactMe, UnitID,StateID from Defect "  30
            ."where (DefectID = \'$fields{line}\')");
@data=&ora_fetch($csr);
($reaℓid,@tail)=@data;
foreach (@data) {
  tr/+/ /;
  s/\s*$//;
  s/%(..)/pack("c",hex($1))/ge;
  }
($id,$titℓe,$prior,$descr,$hfound,$wfound,$cnt,$unit,$state)=@data;
                                                                   40
##########
#Check the acces right for the defect and display
##########
$csr=&ora_open($ℓda,"select Name,Responsible from Unit where "
            ."(UnitID = \'$unit\')");

($name,$resp)=&ora_fetch($csr);


$csr=&ora_open($ℓda,"select PersonID from Team where ((GroupID =  \'$resp\') "  50
            ."and (PersonID = \'$ENV{'REMOTE_USER'}\'))");
$count=0;
while (@data=&ora_fetch($csr)){$count++;}
if ($count < 1) {
   $resp=~tr/+/ /;
   $resp=~s/%(..)/pack("c",hex($1))/ge;
   print "<CENTER>Permission denied:<BR>You are not member of the "
```

```
      ."<B>$resp</B> group!!";
}
else {
   $name=~tr/+/ /;
   $name=~s/%(..)/pack("c",hex($1))/ge;
   $tmp2 = $unit;
   $tmp2=~tr/+/ /;
   $tmp2=~s/%(..)/pack("c",hex($1))/ge;
   $displ = $name." ( ID : ".$tmp2." )";
   $descr=~ s/\n/<BR>/g;
   $tmp = $Param::RGWarePath."ReplyDfct.cgi";
   print <<END;
<TABLE>
<TR><TD ALIGN=right><B>ID :</B></TD><TD> $id</TD></TR>
<TR><TD ALIGN=right><B>Target :</B></TD><TD>$displ</TD></TR>
<TR><TD ALIGN=right><B>Title :</B></TD><TD>$title</TD></TR>
<TR><TD ALIGN=right><B>State :</B></TD><TD>$state</TD></TR>
<TR><TD ALIGN=right><B>Priority :</B></TD><TD>$prior</TD></TR>
<TR><TD VALIGN=top ALIGN=right><B>Description :</B></TD><TD>$descr</TD></TR>
<TR><TD ALIGN=right><B>How Found :</B></TD><TD>$hfound</TD></TR>
<TR><TD ALIGN=right><B>Where Found :</B></TD><TD>$wfound</TD></TR>
<TR><TD ALIGN=right><B>Contact me at :</B></TD><TD>$cnt</TD></TR>
</TABLE>
<CENTER>
<FORM METHOD=GET ACTION=$tmp>
<INPUT TYPE=hidden NAME=dfctid VALUE="$id">
<INPUT TYPE=hidden NAME=targ VALUE="$unit">
<INPUT TYPE=hidden NAME=targname VALUE="$name">
<INPUT TYPE=hidden NAME=state VALUE="$state">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_ChangeState">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareRetarg">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareLnk">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareVLnk">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareHist">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareModify">
END
   }
&ora_close($csr);
&ora_logoff($lda);
print "</CENTER></FORM></BODY></HTML>";
```

EditLink.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, "../pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
```

```
use Oraperl;
use Param;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs){
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
   }

print "Content-type: text/html\n\n";
print "<HTML><BODY $Param::bgcolor $Param::text>";

$ℓda=&ora_login('sid1','xgillman','xgillman');

##########
#Load Similar attributes from database.
##########
$csr=&ora_open($ℓda,"select DefectID, LnkID, Title, Description, HowFound, "
            ."WhereFound, ContactMe from Defect "
            ."where (DefectID = \'$fields{line}\')");
@data=&ora_fetch($csr);
($reaℓid,$reaℓsim,@tail)=@data;
foreach (@data) {
  tr/+/ /;
  s/\s*$//;
  s/%(..)/pack("c",hex($1))/ge;
  }
($id,$sim,$titℓe,$descr,$hfound,$wfound,$cnt)=@data;

##########
#Load the state, priority and unit from the referenced defect.
##########
$csr=&ora_open($ℓda,"select StateID,Priority,UnitID from Defect "
            ."where (DefectID = \'$realsim\')");
@data=&ora_fetch($csr);
foreach (@data) {
  tr/+/ /;
  s/\s*$//;
  s/%(..)/pack("c",hex($1))/ge;
  }
($state,$prior,$unit)=@data;

##########
#Retrieve the name of the targetted unit from the database
##########
$csr=&ora_open($ℓda,"select Name from Unit where (UnitID = \'$unit\')");
($tmp)=&ora_fetch($csr);
$tmp=~tr/+/ /;
$tmp=~s/%(..)/pack("c",hex($1))/ge;
```

```
$displ = $tmp." ( ID : ".$unit." )";

&ora_close($csr);
&ora_logoff($lda);
$tmp = $Param::RGWarePath."ReplyLnk.cgi";
$descr=~ s/\n/\n                    /g;
print <<END;
<TABLE>
<TR><TD ALIGN=right><B>ID :</B></TD><TD>$id</TD></TR>
<TR><TD ALIGN= right><B>Title : </B></TD><TD>$title</TD></TR>
<TR><TD ALIGN= right><B>Similar As : </B></TD><TD>$sim</TD></TR>
<TR><TD ALIGN= right><U>Target</U> : </TD><TD>$displ</TD></TR>
<TR><TD ALIGN= right><U>State</U> : </TD><TD>$state</TD></TR>
<TR><TD ALIGN= right><U>Priority</U> : </TD><TD>$prior</TD></TR>
<TR><TD ALIGN= right><B>Description : </B></TD><TD>$descr</TD></TR>
<TR><TD ALIGN= right><B>How Found : </B></TD><TD>$hfound</TD></TR>
<TR><TD ALIGN= right><B>Where Found : </B></TD><TD>$wfound</TD></TR>
<TR><TD ALIGN= right><B>Contact me at : </B></TD><TD>$cnt</TD></TR>
</TABLE><CENTER>
<FORM METHOD=POST ACTION=$tmp>
<INPUT TYPE=hidden NAME=dfctid VALUE="$id">
<INPUT TYPE=hidden NAME=targ VALUE="$unit">
<INPUT TYPE=hidden NAME=prior VALUE="$prior">
<INPUT TYPE=hidden NAME=state VALUE="$state">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareULnk">
<INPUT TYPE=submit NAME=action VALUE="$Param::btn_GWareHist">
</CENTER></FORM></BODY></HTML>
END
```

LinkHist.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        }
use Param;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fields{$key}=$content;
        }

print "Content-type: text/html\n\n";
print "<HTML><BODY $Param::bgcolor $Param::text>";

$file = $Param::DfctHistPath.$fields{line}.".lnk";
open (HIST, "<$file");
while (<HIST>){
```

```
        tr/+/ /;
        s/%0D%0A/<BR>/g;
        s/%3C/&lt;/g;
        s/%(..)/pack("c",hex($1))/ge;
        print "$_ <BR>";
        }
close HIST;
print "</BODY></HTML>";
```

LinkTo.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "../pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use DisplaySet;
use Time::Local;
use Shell;
use ReportForUnit;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        $fieℓds{$key}=~s/%3C/&lt/g;
        }

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

$ℓda=&ora_login('sid1','xgillman','xgillman');

##########
#Update Database
##########
$csr=&ora_open($ℓda,"update Defect set LnkID =\'$fields{line}\' where "
            ."(DefectID = \'$fields{param}\')");
$csr=&ora_open($ℓda,"select UnitID  from Defect "
            ."where (DefectID = \'$fields{param}\')");
($targ)=&ora_fetch($csr);
&ora_close($csr);
&ora_logoff($ℓda);
```

```
##########
#Update the log file  of the defect
##########
$file = $Param::DfctHistPath."$fields{param}";
open (HIST, ">>$file");
$time = localtime();
$tmp = $fields{line};
$tmp=~tr/+/ /;
$tmp=~s/%(..)/pack("c",hex($1))/ge;

print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect <U>associated</U> on <B>"
        ."$time</B> to <B>$tmp</B></FONT>\n";
print HIST "<B>Comment :<B>\n<I>$fields{comment}</I>\n" if ($fields{comment});
close HIST;

##########
#Display the report of the targetted unit
##########
&ReportForUnit::display($targ);
print "</BODY></HTML>";
```

ReplyDfct.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }
use Env;
use Oraperl;
use Param;
use DisplaySet;
use UnitHierarchy;

$temp=$ENV{'QUERY_STRING'};
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fields{$key}=$content;
}

$id = $fields{dfctid};
$state = $fields{state};

foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $content=~tr/+/ /;
   $content=~s/%(..)/pack("c",hex($1))/ge;
   $fields{$key}=$content;
```

```
}

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########
##########
#Display the form to change the state of the defect in its life cycle
##########
##########
if ($fieℓds{action} eq $Param::btn_ChangeState){
   $tmp = $Param::RGWarePath."ChangeStt.cgi";
   print <<END;
<CENTER>
<FONT COLOR=$Param::PromptColor><B>Changing the State of a Defect.</B></FONT>
<BR><BR><FORM METHOD=GET ACTION=$tmp>
<INPUT TYPE=hidden NAME=dfctid VALUE=$fieℓds{dfctid}>
<INPUT TYPE=hidden NAME=targ VALUE=$fieℓds{targ}>
<INPUT TYPE=hidden NAME=targname VALUE=$fieℓds{targname}>
<TABLE>
<TR><TD VALIGN=top ALIGN=right>New State :</TD>
<TD><SELECT NAME=newstate SIZE=1>
END
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select StateID, Name  from State "
               ."where not (StateID = \'$state')");
   while (($SttID,$SttName)=&ora_fetch($csr)){
      $SttName=~tr/+/ /;
      $SttName=~s/%(..)/pack("c",hex($1))/ge;
      $SttID=~tr/+/ /;
      $SttID=~s/%(..)/pack("c",hex($1))/ge;
      print "<OPTION VALUE=$SttID>$SttName (ID: $SttID)";
   }
   &ora_close($csr);
   &ora_logoff($ℓda);
   print <<END;
</SELECT></TD></TR>
<TR><TD VALIGN=top ALIGN=right>Comment :</TD>
<TD><TEXTAREA NAME=comment COLS=40 ROWS=2 ></TEXTAREA></TD></TR></TABLE>
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_NewState\">
<INPUT TYPE=reset NAME=action VALUE=Reset>
</FORM></CENTER>
END
}

##########
##########
#Display the log file of the defect
##########
```

```
##########
elsif ($fields{action} eq $Param::btn_GWareHist){
  $file = $Param::DfctHistPath.$id;                                    80
  open (HIST, "<$file");
  while (<HIST>){
    tr/+/ /;
    s/%0D%0A/<BR>/g;
    s/%3C/&lt;/g;
    s/%(..)/pack("c",hex($1))/ge;
    print "$_ <BR>";
    }
  close HIST;
}                                                                      90

##########
##########
#Display the form used to retarget the defect to another unit.
##########
##########
elsif ($fields{action} eq $Param::btn_GWareRetarg){
  print <<END;
<CENTER><TABLE><TR>
<TD><FONT COLOR=$Param::PromptColor><B>Current Target:</B></FONT></TD>   100
<TD><B> $fields{targname}</B> (ID : $fields{targ})</TD></TR>
<TR><TD VALIGN=top ALIGN=right><FONT COLOR=$Param::PromptColor>
<B>New Target:</B></FONT></TD><TD>
END
  &UnitHierarchy::display($Param::RGWarePath."Retarget.cgi?dfctid="
                    ."$fields{dfctid}&org=$fields{targ}&orgname="
                    ."$fields{targname}&","New Target","target.gif");
  print "</TD></TR></TABLE></CENTER>";
}
                                                                       110
##########
##########
#Display the form to link a defect to another defect.
##########
##########
elsif ($fields{action} eq $Param::btn_GWareLnk){
  $lda=&ora_login('sid1','xgillman','xgillman');
  $csr=&ora_open($lda,"select DefectID, Title, Priority, StateID "
            ." from Defect where (not(DefectID = \'$id\')and"
            ." (UnitID in (select UnitID from Defect where"            120
            ." (DefectID = \'$fields{dfctid}\')))and(LnkID is NULL))");
  @col=(DefectID, Title, Priority, StateID);
  print "<CENTER>";
  print "<FONT COLOR=$Param::PromptColor><B>Choose a defect as reference.</B>"
      ."</FONT><BR><BR>";
  &DisplaySet::display($csr,\@col,$Param::RGWarePath."LinkTo.cgi",undef,
                  undef,$fields{dfctid});
  print "</CENTER>";
```

```
  &ora_close($csr);
  &ora_logoff($lda);                                                    130
}

##########
##########
#Display the form used to correct defect attributes
#########
##########
elsif ($fields{action} eq $Param::btn_GWareModify){
  $lda=&ora_login('sid1','xgillman','xgillman');
  $csr=&ora_open($lda,"select DefectID, Title, Priority, UnitID, StateID, "   140
            ."Description, HowFound, WhereFound, ContactMe "
            ."from Defect where (DefectID =  \'$fields{dfctid}\')");
  @data=&ora_fetch($csr);
  foreach (@data) {
    tr/+/ /;
    s/%(..)/pack("c",hex($1))/ge;
    s/\s*$//;
    }
  ($id,$title,$prior,$targ,$state,$descr,$hfound,$wfound,$cnt)=@data;
  &ora_close($csr);                                                     150
  &ora_logoff($lda);
  $tmp = $Param::RGWarePath."Correct.cgi";
  print <<END;
<FORM METHOD=POST ACTION=$tmp>
<CENTER>
<INPUT TYPE=hidden NAME=dfctid VALUE=$fields{dfctid}>
<INPUT TYPE=hidden NAME=targ VALUE=$fields{targ}>
<INPUT TYPE=hidden NAME=targname VALUE=\"$fields{targname}\">
<TABLE><TR><TD ALIGN=right>Title :</TD>
<TD><INPUT TYPE=text NAME=title VALUE=\"$title\" MAXLENGTH=40 SIZE=40 ></TD>   160
</TR>
<TR><TD ALIGN=right>Priority :</TD>
<TD><INPUT TYPE=text NAME=prior VALUE=\"$prior\" MAXLENGTH=1 SIZE=1 ></TD></TR>
<TR><TD VALIGN=top ALIGN=right>Description :</TD>
<TD><TEXTAREA NAME=descr COLS=38 ROWS=4 >$descr</TEXTAREA></TD></TR>
<TR><TD ALIGN=right>How Found :</TD>
<TD><INPUT TYPE=text NAME=hfnd  VALUE=\"$hfound\" MAXLENGTH=40 SIZE=40 ></TD>
</TR>
<TR><TD ALIGN=right>Where Found :</TD>
<TD><INPUT TYPE=text NAME=wfnd  VALUE=\"$wfound\" MAXLENGTH=40 SIZE=40 ></TD>   170
</TR>
<TR><TD ALIGN=right>Contact me at :</TD>
<TD><INPUT TYPE=text NAME=cnt  VALUE=\"$cnt\" MAXLENGTH=40 SIZE=40 ></TD></TR>
<TR><TD VALIGN=top ALIGN=right>Comment :</TD>
<TD><TEXTAREA NAME=comment COLS=40 ROWS=2 ></TEXTAREA></TD></TR>
</TABLE><INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_Cor\">
</CENTER></FORM>
END
}
```

```
##########
##########
#Display the summary of every similar of the defect
##########
##########
elsif ($fields{action} eq $Param::btn_GWareVLnk){
  print "<CENTER>";
  print "<FONT COLOR=$Param::PromptColor><B>Defect Associated to Defect "
      ."\"$fields{dfctid}\".</B></FONT><BR><BR>";
  $lda=&ora_login('sid1','xgillman','xgillman');
  $count=0;
  $cond="(LnkID=\'$id\')";
  $csr=&ora_open($lda,"select DefectID from Defect where $cond");
  while (@data=&ora_fetch($csr)){
    $count++;
  }
  $comp=$count;
  $enter=1;
  while (($enter > 0)||($comp > $count)){
    $enter=0;
    $count=$comp;
    $comp=0;
    $cond=$cond." or (LnkID in (select DefectID from Defect where $cond))";
    $csr=&ora_open($lda,"select DefectID,Title from Defect where ($cond)");
    while (@data=&ora_fetch($csr)){
      $comp++;
    }
  }
  $csr=&ora_open($lda,"select DefectID,Title from Defect where ($cond)");
  @col=(DefectID, Title);
  &DisplaySet::display($csr,\@col,$Param::RGWarePath."EditLink.cgi");
  print "</CENTER>";
  &ora_close($csr);
  &ora_logoff($lda);
}

print "</BODY></HTML>";
```

ReplyLnk.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, "../pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
use Param;
```

```
use DisplaySet;                                                          10
use ReportForUnit;
use Time::Local;
use Shell;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fields{$key}=$content;
}                                                                        20

$fields{action}=~tr/+/ /;
$fields{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########                                                               30
##########
#View the log file of the similar.
##########
##########
if ($fields{action} eq "$Param::btn_GWareHist") {
   $file = $Param::DfctHistPath.$fields{dfctid};
   open (HIST, "<$file");
   while (<HIST>){
      tr/+/ /;
      s/%0D%0A/<BR>/g;                                                   40
      s/%3C/&lt;/g;
      s/%(..)/pack("c",hex($1))/ge;
      print "$_ <BR>";
      }
   close HIST;
   }

##########
##########
#Dissociate a similar from its refernce.                                 50
##########
##########
else { # $fields{action} eq "$Param::btn_GWareULnk"

##########
#Update the Defect table.
##########
   $lda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($lda,"update Defect set StateID = \'$fields{state}\', "
              ."UnitID =\'$fields{targ}\', Priority=\'$fields{prior}\',"  60
```

```
                ."LnkID=NULL where (DefectID=\'$fields{dfctid}\')");
   &ora_close($csr);
   &ora_logoff($lda);

##########
#Update the log file of the defect.
##########
   $file = $Param::DfctHistPath."$fields{dfctid}";
   open (HIST, ">>$file");
   $time = localtime();

   print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect <U>disociated</U>"
           ." on <B>$time</B></FONT>\n";
   close HIST;

##########
#Display the report for the targetted unit.
##########
   &ReportForUnit::display($fields{targ});
   }
print "</BODY></HTML>";
```

ReportForUnit.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "lib/");
        }

use ReportForUnit;

$temp = $ENV{'QUERY_STRING'};
@pairs=split(/&/,$temp);
foreach $item(@pairs){
        ($key,$content)=split(/=/,$item,3);
        $fields{$key}=$content;
        $fields{$key}=~tr/+/ /;
        $fields{$key}=~s/%(..)/pack("c",hex($1))/ge;
        }
print "Content-type: text/html\n\n";
&ReportForUnit::display($fields{UnitID},$fields{UnitName});
```

Retarget.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "../pub/lib/");
        unshift (@INC, "lib/");
        }
```

```
use Env;
use Oraperl;
use Param;
use Email;
use DisplaySet;
use Time::Local;
use ReportForUnit;

$temp=$ENV{'QUERY_STRING'};
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        }

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

$ℓda=&ora_login('sid1','xgillman','xgillman');

##########
#Update the database table
#########
$csr=&ora_open($ℓda,"update Defect set UnitID = \'$fields{UnitID}\' "
            ."where (DefectID = \'$fields{dfctid}\')");

##########
#Update the defect log file
##########
$fiℓe = $Param::DfctHistPath.$fieℓds{dfctid};
$time = localtime();
$tmp = $fieℓds{UnitID};
$tmp=~tr/+/ /;
$tmp=~s/%(..)/pack("c",hex($1))/ge;

open (HIST, ">>$file");
print HIST "<FONT SIZE=4 COLOR=#0000FF>--- Defect <U>retargetted</U>"
        ." on <B>$time</B></FONT><BR><B>New Target :</B> $tmp<BR>";
close HIST;

##########
#Send a notification e-mail to unit responsibles
##########
if ($Param::UserMailWarnin) {&Email::email($ℓda,$tmp)}
&ora_close($csr);
&ora_logoff($ℓda);

##########
```

```
#Display the report of the unit.
##########
$fields{org}=~tr/+/ /;
$fields{org}=~s/%(..)/pack("c",hex($1))/ge;                                    60
$fields{orgname}=~tr/+/ /;
$fields{orgname}=~s/%(..)/pack("c",hex($1))/ge;
&ReportForUnit::display($fields{org},$fields{orgname});
print "</CENTER></BODY></HTML>";
```

C.3 "admin" directory

AdminForm.pm

```
package AdminForm;

require 5.000;

BEGIN {
        unshift (@INC, ". ./pub/lib/");
        }

use Param;

sub display {
$tmp = $Param::RAdminPath."Admin.cgi";
print <<END;
<FORM METHOD=\"POST\" ACTION=\"$tmp\"
<CENTER>
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminPerson\">
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminGroup\">
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminUnit\">
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminDfct\">
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminLnk\">
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_AdminState\">
</CENTER></FORM>
END
}
```

EditGrp.pm

```
package EditGrp;

require 5.000;

BEGIN {
        unshift (@INC, ". ./. ./pub/lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

sub display {
    my ($grp)= @_;

    $tmp = $Param::RAdminPath."ReplyGrp.cgi";
```

```
    $grpname = $grp;
    $grpname=~tr/+/ /;
    $grpname=~s/%(..)/pack("c",hex($1))/ge;
    print "<CENTER><B>GroupID :</B> $grpname<BR>";

    print "<B>Current Members of the Group :</B><BR>";
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"select PersonID  from Team where (GroupID=\'$grp\')");
    @col=(PersonID);
    &DisplaySet::display($csr,\@col);

    print "<FORM METHOD=POST ACTION=$tmp>";
    print "<INPUT TYPE=hidden NAME=grpname VALUE=\"$grpname\">";
    print "<B>Delete the Group from the Database :</B><BR>";
    print "<INPUT TYPE=submit NAME=action VALUE=\"Delete\">";
    print "</FORM>";

    $csr=&ora_open($ℓda,"select PersonID  from Person");
    @col=(PersonID);
    print "<B>Change the Composition of the Group :</B><BR>";
    print "Click on a PersonID. If that profile is already in the group,"
        ." it will be removed otherwize it will be added.<BR>";
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."ReplyGrp.cgi",
                         undef,undef,$grp);
    print "</CENTER>";
    &ora_close($csr);
    &ora_logoff($ℓda);
}
```

Admin.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        }
```

```
$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML>
<CENTER>
END

##########
##########
#Display the personal profiles for administration.
##########
##########
if ($fieℓds{action} eq $Param::btn_AdminPerson){
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select PersonID, LastName, FirstName, Email,"
              ." Usr_Access, Adm_Access  from Person");
   print "<FONT COLOR=$Param::PromptColor><B>Person Table</B></FONT><BR>";
   @col=(PersonID, LastName, FirstName, Email, Usr_Access, Adm_Access);
   &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblPrs.cgi",
                        $Param::btn_Create);
   &ora_close($csr);
   &ora_logoff($ℓda);
}

##########
##########
#Display the Unit table for administration.
##########
##########
elsif ($fieℓds{action} eq $Param::btn_AdminUnit) {
   print "<FONT COLOR=$Param::PromptColor><B>Unit Table</B></FONT> <BR>";
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select UnitID, Name, Release, SubUnitOf,"
              ." Responsible  from Unit");
   @col=(UnitID, Name, Release, SubUnitOf, Responsible);
   &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblUnt.cgi",
                        $Param::btn_Create);
   &ora_close($csr);
   &ora_logoff($ℓda);
}

##########
##########
#Display the group table for administration.
##########
##########
elsif ($fieℓds{action} eq $Param::btn_AdminGroup) {
   print "<FONT COLOR=$Param::PromptColor><B>Group Table.</B></FONT><BR>";
```

```
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"select GroupID from GroupPers");
    @col=(GroupID);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblGrp.cgi",
                         $Param::btn_Create);
    &ora_close($csr);
    &ora_logoff($ℓda);
}

##########
##########
#Display the defect table (only unlinked defects) for administration.
##########
##########
elsif ($fieℓds{action} eq $Param::btn_AdminDfct){
    print "<FONT COLOR=$Param::PromptColor><B>Defect Table.</B></FONT><BR>";
    print "<FONT COLOR=$Param::PromptColor><B>Click on a Defect to "
        ."<FONT COLOR=#FF0000>DELETE</FONT> it!</B></FONT><BR>";
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"select DefectID, Title, Priority, UnitID, StateID "
                  ."from Defect where (LnkID is NULL)");
    @col=(DefectID, Title, Priority, UnitID, StateID);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblDfct.cgi",
                         $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
}

##########
##########
#Display the defect table (only similars) administration.
##########
##########
elsif ($fieℓds{action} eq $Param::btn_AdminLnk){
    print "<FONT COLOR=$Param::PromptColor><B>Similars Table.</B></FONT><BR>";
    print "<FONT COLOR=$Param::PromptColor><B>Click on a Item to "
        ."<FONT COLOR=#FF0000>DELETE</FONT> it!</B></FONT><BR>";
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"select DefectID, Title, LnkID  from Defect "
                  ."where (LnkID is not NULL)");
    @col=(DefectID, Title, LnkID);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblLnk.cgi",
                         $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
}

##########
##########
#Display the Sate table for administration (of the defect life cycle).
##########
```

```
##########
elsif ($fieℓds{action} eq $Param::btn_AdminState){
   print "<FONT COLOR=$Param::PromptColor><B>Defect Lifecycle Table."
        ."</B></FONT><BR>";
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select StateID, Name from State");
   @col=(StateID, Name);
   &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblStt.cgi",
                        $Param::btn_Create);
   &ora_close($csr);                                                    130
   &ora_logoff($ℓda);
}

##########
#Display the administrator menu.
##########
print "<BR><BR>";
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

GroupCreated.cgi

```
#!/usr/local/bin/perl
BEGIN {
       unshift (@INC, ". ./pub/lib/");
       unshift (@INC, "lib/");
       }

use Env;
use Oraperl;
use Param;
use AdminForm;                                                          10
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
   $fieℓds{$key}=~s/%3C/&lt/g;
}
                                                                        20
##########
#Add the group to the table.
##########
$ℓda=&ora_login('sid1','xgillman','xgillman');
$csr=&ora_open($ℓda,"insert into GroupPers (GroupID) values"
            ." (\'$fields{grpname}\')");

print <<END;
```

```
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text><CENTER>\n
<FONT COLOR=$Param::PromptColor><B>New group commited.<BR>
Group Table.</B></FONT><BR>
END

##########
#Display the group table for administration.
##########
$csr=&ora_open($ℓda,'select GroupID from GroupPers');
@col=(GroupID);
&DisplaySet::display($csr,\@col,$Param::RAdminPath."TblGrp.cgi",
                $Param::btn_Create,$Param::btn_Reset);
&ora_close($csr);
&ora_logoff($ℓda);

##########
#Display the administrator menu.
##########
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

PersonCreated.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, ". ./pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;
use Time::Local;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
  ($key,$content)=split(/=/,$item,3);
  $fieℓds{$key}=$content;
  $fieℓds{$key}=~s/%3C/&lt/g;
  }

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END
```

```
##########
#Update the Person table of the database.
##########
$fieℓds{usr_access}="off" if !($fieℓds{usr_access});
$fieℓds{adm_access}="off" if !($fieℓds{adm_access});

$ℓda=&ora_login('sid1','xgillman','xgillman');
$csr=&ora_open($ℓda,"insert into Person (PersonID , FirstName, LastName,"
            ."Email,Usr_Access,Adm_Access) values (\'$fields{prsid}\'"
            .",\'$fields{fname}\',\'$fields{lname}\',"
            ."\'$fields{email}\',\'$fields{usr_access}\',"
            ."\'$fields{adm_access}\')");

##########
#Display the Person table for administration.
##########
$csr=&ora_open($ℓda,"select PersonID, LastName, FirstName, Email, Usr_Access,"
            ." Adm_Access  from Person");
@col=(PersonID, LastName, FirstName, Email, Usr_Access, Adm_Access);
print "<FONT COLOR=$Param::PromptColor><B>Person Table</B></FONT> <BR>";
&DisplaySet::display($csr,\@col,$Param::RAdminPath."TblPrs.cgi",
                    $Param::btn_Create);
&ora_close($csr);
&ora_logoff($ℓda);
print "<B><FONT COLOR=$Param::PromptColor>New profile commited :</FONT>"
    ."<FONT COLOR=#000000>$fields{prsid}</FONT></B><BR>\n";

##########
#Update the .htpasswd and .htgrp files (access right files).
#Note the files have to be on "nobody" access right !!!!
##########
($rnd1,$rnd2,@tail) = localtime();
srand($rnd1);
$i = rand 255;
$i =~ s/^(..+)\..+$/pack("c",hex($1))/e;
srand ($rnd2);
$j =rand 255;
$j =~ s/^(..+)\..+$/pack("c",hex($1))/e;
$pwd = crypt($fieℓds{passwd},$i.$j);

$fiℓe = "$Param::DTPath".".htpasswd";
open (PSWD, ">> $file");
print PSWD "$fields{prsid}:$pwd\n";
close(PSWD);

if ($fieℓds{usr_access}){
    $fiℓe = "$Param::DTPath".".htgrp";
    open (GRP, "< $file");
    $ℓine1 = <GRP>;
    $ℓine2 = <GRP>;
```

```
    close(GRP);
    open (GRP,"> $file");
    $line2=~s/\n$//;                                                    80
    print GRP $line1;
    print GRP $line2." ".$fields{prsid};
    close (GRP);
    }
if ($fields{adm_access}){
    $file = "$Param::DTPath".".htgrp";
    open (GRP, "< $file");
    $line1 = <GRP>;
    $line2 = <GRP>;
    close(GRP);                                                         90
    open (GRP,"> $file");
    $line1=~s/\n$//;
    print GRP $line1." ".$fields{prsid}."\n";
    print GRP $line2;
    close (GRP);
    }

##########
#Display the administrator menu.
##########                                                             100
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

ReplyGrp.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;                                                          10
use EditGrp;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fields{$key}=$content;
}
                                                                        20
$fields{action}=~tr/+/ /;
$fields{action}=~s/%(..)/pack("c",hex($1))/ge;
```

```
print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>\n
<CENTER>
END
$ℓda=&ora_login('sid1','xgillman','xgillman');

##########
#Try to delete the group from the database.
##########
if ($fieℓds{action} eq "Delete") {
   $csr=&ora_open($ℓda,"delete from Team  "
               ."where (GroupID = \'$fields{grpname}\')");
   $csr=&ora_open($ℓda,"delete from GroupPers  where "
               ."(GroupID = \'$fields{grpname}\')");

##########
#Display the group table for administration.
##########
   $csr=&ora_open($ℓda,"select GroupID from GroupPers");
   print "<FONT COLOR=$Param::PromptColor><B>Group Deleted</B></FONT> <BR>\n";
   @col=(GroupID);
   &DisplaySet::display($csr, \@col,$Param::RAdminPath."TblGrp.cgi",
                        $Param::btn_Create,$Param::btn_Reset);
   &ora_close($csr);
   &ora_logoff($ℓda);
  &AdminForm::display();
}

##########
#Change the composition of the edited group (from Team table).
#i.e. the user selected a peronal profile.
##########
else {
   $csr=&ora_open($ℓda,"select PersonID from Team "
               ." where ((GroupID=\'$fields{param}\')"
               ." and (PersonID=\'$fields{line}\'))");
   $count=0;
   while (@data=&ora_fetch($csr)){ $count++}
   if ($count == 0) {

##########
#The profile is not member of the group: the user wants to add it.
#########
      $csr=&ora_open ($ℓda,"insert into Team (GroupID, PersonID )"
                  ." values (\'$fields{param}\',\'$fields{line}\')");
      $fieℓds{param}=~tr/+/ /;
      $fieℓds{param}=~s/%3C/&lt/g;
      $fieℓds{param}=~s/%(..)/pack("c",hex($1))/ge;
      $fieℓds{line}=~tr/+/ /;
```

```
    $fields{param}=~s/%3C/&lt/g;
    $fields{line}=~s/%(..)/pack("c",hex($1))/ge;
    print "<FONT COLOR=$Param::PromptColor><B>Inserted profile "
        ."<FONT COLOR=#000000>$fields{line}</FONT> in group "
        ."<FONT COLOR=#000000>$fields{param}</FONT></FONT><BR>";
  }
  else {

##########
#The profile is member of the group: the user wants to remove it.
#########
    $csr=&ora_open ($lda,"delete from Team "
                ."where ((GroupID=\'$fields{param}\')"
                ." and (PersonID = \'$fields{line}\'))");
    $fields{param}=~tr/+/ /;
    $fields{param}=~s/%(..)/pack("c",hex($1))/ge;
    $fields{param}=~s/%3C/&lt/g;
    $fields{line}=~tr/+/ /;
    $fields{param}=~s/%3C/&lt/g;
    $fields{line}=~s/%(..)/pack("c",hex($1))/ge;
    print "<FONT COLOR=$Param::PromptColor><B>Deleted profile "
        ."<FONT COLOR=#000000>$fields{line}</FONT> from group "
        ."<FONT COLOR=#000000>$fields{param}</B></FONT></FONT><BR>";
  }
  &ora_close($csr);
  &ora_logoff($lda);

##########
#Display the group composition.
#########
  &EditGrp::display($fields{param});

##########
#Display the administrator menu.
#########
  &AdminForm::display();
}

print "</CENTER></BODY></HTML>";
```

ReplyPrs.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, "../pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
```

```
use Param;
use AdminForm;
use DisplaySet;
use Shell;
use Time::Local;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
   $fieℓds{$key}=~s/%3C/&lt/g;
}

$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END

##########
##########
#Correct the edited person profile.
##########
##########
if ($fieℓds{action} eq $Param::btn_Cor) {
   $fieℓds{usr_access}="off" if !$fieℓds{usr_access};
   $fieℓds{adm_access}="off" if !$fieℓds{adm_access};


##########
#Update the database Person table.
##########
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"update Person set FirstName=\'$fields{fname}\',"
                ." LastName=\'$fields{lname}\', Email=\'$fields{email}\',"
                ." Usr_Access=\'$fields{usr_access}\', Adm_Access="
                ."\'$fields{adm_access}\' "
                ."where (PersonID = \'$fields{prsid}\')");
   $csr=&ora_open($ℓda,"select PersonID, LastName, FirstName, Email,"
                ." Usr_Access, Adm_Access  from Person");
   @col=(PersonID, LastName, FirstName, Email, Usr_Access, Adm_Access);
   print "<FONT COLOR=$Param::PromptColor><B>Person Table</B></FONT> <BR>";
   &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblPrs.cgi",
                        $Param::btn_Create);
   &ora_close($csr);
   &ora_logoff($ℓda);
```

```
##########                                                              60
#Update the access right files if a new password is given.
##########
  if ($fields{passwd}){
    ($rnd1,$rnd2,@tail) = localtime();
    srand($rnd1);
    $i = rand 255;
    $i =~ s/^(..+)\..+$/pack("c",hex($1))/e;
    srand ($rnd2);
    $j =rand 255;
    $j =~ s/^(..+)\..+$/pack("c",hex($1))/e;                            70
    $pwd = crypt($fields{passwd},$i.$j);

    $file = "$Param::DTPath".".htpasswd";
    open (PSWD, "<$file");
    @tmp = "";
    while (<PSWD>) {
      $_ = $fields{prsid}.":$pwd\n" if (/^$fields{prsid}.*$/);
      push @tmp, $_;
    }
    close(PSWD);                                                        80
    open (PSWD, ">$file");
    foreach (@tmp) {
      print PSWD "$_"if ($_);
    }
    close(PSWD);
  }

  $file = "$Param::DTPath".".htgrp";
  open (GRP, "<$file");
  $admingrp = <GRP>;                                                    90
  $gwaregrp = <GRP>;
  close(GRP);

  ($grp, $members) = split (/:/,$gwaregrp);
  @tmp = split(/\s/,$members);
  undef $members;
  foreach (@tmp) {
    $members = $members." ".$_ if ($_ ne $fields{prsid});
    }
  $members = $members." ".$fields{prsid} if ($fields{usr_access} eq "on");  100
  $members =~ s/^\s(.*)$/$1/;
  $gwaregrp= $grp.":".$members;

  ($grp, $members) = split (/:/,$admingrp);
  @tmp = split(/\s/,$members);
  undef $members;
  foreach (@tmp) {
    $members = $members." ".$_ if ($_ ne $fields{prsid});
  }
  $members = $members." ".$fields{prsid} if ($fields{adm_access} eq "on");  110
```

```
    $members =~ s/^\s(.*)$/$1/;
    $admingrp= $grp.":".$members;

    open (GRP,">$file");
    print GRP "$admingrp\n$gwaregrp";
    close (GRP);


    print "<FONT COLOR=$Param::PromptColor><B>Profile "
        ."<FONT COLOR=#000000>$fields{prsid}</FONT> modified.</B></FONT><BR>";
    &AdminForm::display();
    print "</CENTER>";
}

##########
##########
#Delete the edited person profile.
##########
##########
else {

##########
#Update the database (Person and Team tables).
##########
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from Team "
                ."where (PersonID = \'$fields{prsid}\')");
    $csr=&ora_open($ℓda,"delete from Person"
                ." where (PersonID=\'$fields{prsid}\')");
    $csr=&ora_open($ℓda,"select PersonID, LastName, FirstName, Email,"
                ." Usr_Access, Adm_Access  from Person");
    @col=(PersonID, LastName, FirstName, Email, Usr_Access, Adm_Access);
    print "<FONT COLOR=$Param::PromptColor><B>Person Table</B></FONT><BR>\n";
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblPrs.cgi",
                        $Param::btn_Create);
    &ora_close($csr);
    &ora_logoff($ℓda);

##########
#Update the right access files (.htpasswd and .htgrp).
##########
    $fiℓe = "$Param::DTPath".".htpasswd";
    open (PSWD, "<$file");
    @tmp = "";
    while (<PSWD>) {
      push @tmp, $_ if !(/^$fieℓds{prsid}.*$/);
    }
    close(PSWD);
    open (PSWD, ">$file");
    foreach (@tmp) {
      print PSWD "$_"if ($_);
```

```
    }
    close(PSWD);
    $file = "$Param::DTPath".".htgrp";
    open (GRP, "< $file");
    $admingrp = <GRP>;
    $gwaregrp = <GRP>;
    close(GRP);

    ($grp, $members) = split (/:/,$admingrp);                                    170
    @tmp = split(/\s/,$members);
    undef $members;
    foreach (@tmp) {
      $members = $members." ".$_ if ($_ ne $fields{prsid});
    }
    $members =~ s/^\s(.*)$/$1/;
    $admingrp= $grp.":".$members;

    ($grp, $members) = split (/:/,$gwaregrp);
    @tmp = split(/\s/,$members);                                                 180
    undef $members;
    foreach (@tmp) {
      $members = $members." ".$_ if ($_ ne $fields{prsid});
    }
    $members =~ s/^\s(.*)$/$1/;
    $gwaregrp= $grp.":".$members;

    open (GRP,">$file");
    print GRP "$admingrp\n$gwaregrp";
    close (GRP);                                                                 190

    print "<CENTER>";

##########
#Display the administrator menu.
##########
    &AdminForm::display();
    print "</CENTER>";
}
print "</BODY></HTML>";                                                          200
```

ReplyStt.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
```

```
use Param;
use AdminForm;
use DisplaySet;
use Shell;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
}

$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END

##########
##########
#Correct the edited state entity.
##########
##########
if ($fieℓds{action} eq $Param::btn_Cor) {
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"update State set Name=\'$fields{name}\' "
                 ."where (StateID = \'$fields{state}\')");
    $csr=&ora_open($ℓda,'select StateID, Name  from State');
    @col=(StateID, Name);
    print "<FONT COLOR=$Param::PromptColor><B>State  "
        ."<FONT COLOR=#000000>$fields{state}</FONT> modified.</B></FONT><BR>";
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblStt.cgi",
                         $Param::btn_Create,$Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
    &AdminForm::display();
    }

##########
##########
#Delete the edited state entity.
##########
##########
else {
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from State "
                 ."where (StateID=\'$fields{state}\')");
    $csr=&ora_open($ℓda,'select StateID, Name  from State');
```

```
    @col=(StateID, Name);
    print "<FONT COLOR=$Param::PromptColor><B>State "
        ."<FONT COLOR=#000000>$fields{state}</FONT> deleted.</B></FONT><BR>";
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblStt.cgi",
                        $Param::btn_Create,$Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($lda);
    &AdminForm::display();
    }

print "</CENTER></BODY></HTML>";
```

ReplyUnt.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fields{$key}=$content;
        }

$fields{action}=~tr/+/ /;
$fields{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END

##########
##########
#Correct the edited unit entity.
##########
##########
if ($fields{action} eq $Param::btn_Cor) {
    $lda=&ora_login('sid1','xgillman','xgillman');
    if ($fields{subunt} eq "none") {
```

```
        $csr=&ora_open($ℓda,"update Unit set Name=\'$fields{name}\',"
                  ." Release=\'$fields{rel}\', Responsible=\'$fields{resp}\'"
                  ", SubUnitOf=NULL"
                  ." where (UnitID = \'$fields{unitid}\')");
      }
    else {
        $csr=&ora_open($ℓda,"update Unit set Name=\'$fields{name}\', "
                  ."Release=\'$fields{rel}\',Responsible=\'$fields{resp}\',"
                  ."SubUnitOf=\'$fields{subunt}\'"
                  ." where (UnitID = \'$fields{unitid}\')");
        }
    print "<FONT COLOR=$Param::PromptColor><B>Unit "
        ."<FONT COLOR=#000000>$fields{unitid}</FONT> modified.</B></FONT><BR>";
    $csr=&ora_open($ℓda,"select UnitID, Name, Release, SubUnitOf, Responsible "
                ." from Unit");
    @col=(UnitID, Name, Release, SubUnitOf, Responsible);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblUnt.cgi",
                        $Param::btn_Create);
    &ora_close($csr);
    &ora_logoff($ℓda);
    &AdminForm::display();
    }

##########
##########
#Delete the edited unit entity.
##########
##########
else {
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from Unit where (UnitID=\'$fields{unitid}\')");
    print "<FONT COLOR=$Param::PromptColor><B>Unit "
        ."<FONT COLOR=#000000>$fields{unitid}</FONT> deleted.</B></FONT><BR>";
    $csr=&ora_open($ℓda,'select UnitID, Name, Release, SubUnitOf, Responsible "
                ." from Unit');
    @col=(UnitID, Name, Release, SubUnitOf, Responsible);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblUnt.cgi",
                        $Param::btn_Create);
    &AdminForm::display();
    &ora_close($csr);
    &ora_logoff($ℓda);
    }
print "</CENTER></BODY></HTML>";
```

StateCreated.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
```

```
        }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
   $fieℓds{$key}=~s/%3C/&lt/g;
}

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>\n
<CENTER>
END

##########
#Create the new state record in the database.
##########
$ℓda=&ora_login('sid1','xgillman','xgillman');
$csr=&ora_open($ℓda,"insert into State (StateID, Name) values "
             ."(\'$fields{state}\', \'$fields{name}\')");
$csr=&ora_open($ℓda,'select StateID, Name  from State');
@col=(StateID, Name);
print "<FONT COLOR=$Param::PromptColor><B>New state commited.</B></FONT><BR>";
&DisplaySet::display($csr,\@col,$Param::RAdminPath."TblStt.cgi",
                  $Param::btn_Create,$Param::btn_Reset);
&ora_close($csr);
&ora_logoff($ℓda);
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

TblDfct.cgi

```
#!/usr/local/bin/perl
BEGIN {
       unshift (@INC, ". ./pub/lib/");
       unshift (@INC, "lib/");
       }

use Env;
use Oraperl;
use Param;
use AdminForm;
```

```
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
}


$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>\n
<CENTER>
END

##########
##########
#Delete all defects (and similars) from the database.
##########
##########
if ($fieℓds{action} eq $Param::btn_Reset){
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from Defect where (LnkID is not NULL)");
    $csr=&ora_open($ℓda,"delete from Defect ");
    print "<FONT COLOR=$Param::PromptColor><B>Defect Table.</B></FONT><BR>";
    $csr=&ora_open($ℓda,"select DefectID, Title, Priority, UnitID, StateID"
                 ." from Defect");
    @col=(DefectID, Title, Priority, UnitID, State);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblDfct.cgi",
                         $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
    }

##########
##########
#Delete the selected defect from the database.
#Note no foreign key violation is handled.
##########
##########
else {
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from Defect "
                  ."where (DefectID=\'$fields{line}\')");
    print "<FONT COLOR=$Param::PromptColor><B>Defect Table.</B></FONT><BR>";
    $csr=&ora_open($ℓda,"select DefectID, Title, Priority, UnitID, StateID"
                 ." from Defect");
```

```
    @col=(DefectID, Title, Priority, UnitID, State);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblDfct.cgi",
                         $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
    }
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

TblGrp.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "../pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;                                                    10
use EditGrp;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        }
                                                                  20

$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

###########                                                       30
#Display the form used to create a new group.
###########
if ($fieℓds{action} eq $Param::btn_Create){
    $tmp = $Param::RAdminPath."GroupCreated.cgi";
    print "<FORM METHOD=\"POST\" ACTION=$tmp>";
    print "<CENTER>";
    print "Name : <INPUT TYPE=text NAME=grpname MAXLENGHT=30 SIZE=30><BR>";
    print "<INPUT TYPE=submit VALUE=Commit><INPUT TYPE=reset VALUE=Reset>\n";
    print "</CENTER></FORM>";
```

```
    }                                                                    40

############
#Display the selected group composition.
############
else {
    &EditGrp::display($fieℓds{line});
    }
print "</BODY></HTML>";
```

TblLnk.cgi

```
#!/usr/local/bin/perl

BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;                                                               10
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fieℓds{$key}=$content;
        }
                                                                         20
$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END

##########                                                               30
##########
#Delete all similars from the database.
##########
##########
if ($fieℓds{action} eq $Param::btn_Reset){

    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from defect where (LnkID is not NULL)");
```

```
    print "<FONT COLOR=$Param::PromptColor><B>Similar Table.</B></FONT><BR>";
    $csr=&ora_open($ℓda,"select DefectID, LnkID from Defect"                    40
                ." where (LnkID is not NULL)");
    @col=(DefectID, Title, LnkID);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblLnk.cgi",
                        $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
    }

##########
##########                                                                    50
#Delete the selected similar from the database.
##########
##########
else {
    $ℓda=&ora_login('sid1','xgillman','xgillman');
    $csr=&ora_open($ℓda,"delete from Defect "
                ." where (DefectID=\'$fields{line}\')");
    print "<FONT COLOR=$Param::PromptColor><B>Similar Table.</B></FONT><BR>";
    $csr=&ora_open($ℓda,"select DefectID, Title, LnkID  from Defect"
                ." where (LnkID is not NULL)");                                 60
    @col=(DefectID, Title, LnkID);
    &DisplaySet::display($csr,\@col,$Param::RAdminPath."TblLnk.cgi",
                        $Param::btn_Reset);
    &ora_close($csr);
    &ora_logoff($ℓda);
    }
&AdminForm::display();
print "</CENTER></BODY></HTML>";
```

TblPrs.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, ". ./pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;                                                                  10
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fieℓds{$key}=$content;
```

```
}

$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########
##########
#Display the form used to create a new person profile
##########
##########
if ($fieℓds{action} eq $Param::btn_Create){
   $tmp = $Param::RAdminPath. "PersonCreated.cgi";
   print <<END;
<FORM METHOD=POST ACTION=$tmp>
<CENTER>
<FONT COLOR=$Param::PromptColor><B>Creation of a new Profile.</B>
</FONT><BR><BR><TABLE>
<TR><TD ALIGN=right>ID :</TD>
<TD><INPUT TYPE=text NAME=prsid MAXLENGHT=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right>Password :</TD>
<TD><INPUT TYPE=password NAME=passwd MAXLENGHT=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right>First Name :</TD>
<TD><INPUT TYPE=text NAME=fname MAXLENGTH=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right>Last Name :</TD>
<TD><INPUT TYPE=text NAME=lname MAXLENGTH=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right>E-Mail :</TD>
<TD><INPUT TYPE=text NAME=email  MAXLENGTH=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right><FONT COLOR=$Param::PromptColor>
<B>Access Rights:</B></FONT></TD>
<TD>(Every user has automatically the User access right.)</TD></TR>
<TR><TD></TD><TD><INPUT TYPE=checkbox NAME=usr_access>Unit Manager</TD></TR>
<TR><TD></TD><TD><INPUT TYPE=checkbox NAME=adm_access>Admin</TD><TR>
</TABLE>
<INPUT TYPE=submit VALUE=Commit><INPUT TYPE=reset VALUE=Reset>
</CENTER></FORM>
END
}

##########
##########
#Edit the selected person profile.
##########
##########
else {
   $ℓda=&ora_login('sid1','xgillman','xgillman');
```

```
   $csr=&ora_open($ℓda,"select PersonID, LastName, FirstName, Email,"
                  ." Usr_Access, Adm_Access  from Person "
                  ."where (PersonID = \'$fields{line}\')");
   @data = &ora_fetch($csr);
   foreach (@data) {
      tr/+/ /;
      s/[\s]+$//;
      s/%(..)/pack("c",hex($1))/ge;
   }
   ($PersonID,$LastName,$FirstName,$Emaiℓ,$Usr_Access,$Adm_Access)=@data;
   &ora_close($csr);
   &ora_logoff($ℓda);

   $tmp = $Param::RAdminPath."ReplyPrs.cgi";
   print <<END;
<CENTER><FORM METHOD=POST ACTION=$tmp><TABLE>
<TR><TD ALIGN=right>PersonID :</TD>
<TD>$PersonID<INPUT TYPE=hidden NAME=prsid VALUE=\"$PersonID\"></TD></TR>
<TR><TD ALIGN=right>New Password :</TD>
<TD><INPUT TYPE=password NAME=passwd MAXLENGHT=40 SIZE=40></TD></TR>
<TR><TD ALIGN=right>First Name :</TD>
<TD><INPUT TYPE=text NAME=fname VALUE=\"$FirstName\" MAXLENGHT=40 SIZE=40>
</TD></TR>
<TR><TD ALIGN=right>Last Name :</TD>
<TD><INPUT TYPE=text NAME=lname VALUE=\"$LastName\" MAXLENGHT=40 SIZE=40>
</TD></TR>
<TR><TD ALIGN=right>EMail :</TD>
<TD><INPUT TYPE=text NAME=email VALUE=\"$Email\" MAXLENGHT=40 SIZE=40>
</TD></TR>
<TR><TD ALIGN=right><FONT COLOR=$Param::PromptColor>
<B>Access Rights:</B></FONT></TD>
<TD>(Every user has automatically the User access right.)</TD></TR>
<TR><TD></TD><TD><INPUT TYPE=checkbox
END
   print "CHECKED " if ($Usr_Access ne "off");
   print "NAME=usr_access>Unit Manager</TD></TR>";
   print "<TR><TD></TD><TD><INPUT TYPE=checkbox ";
   print "CHECKED " if ($Adm_Access ne "off");
   print "NAME=adm_access>Admin</TD></TR>";
   print "</TABLE><INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_Cor\">";
   print "<INPUT TYPE=reset VALUE=\"$Param::btn_Reset\">";
   print "<INPUT TYPE=submit NAME=action VALUE=\"Delete\"></FORM></CENTER>";
}
print "</BODY></HTML>";
```

TblStt.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, ". ./pub/lib/");
```

```
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
   ($key,$content)=split(/=/,$item,3);
   $fields{$key}=$content;
}

$fields{action}=~tr/+/ /;
$fields{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
END

##########
##########
#Display the form used to create a new state in the defect life cycle
##########
##########
if ($fields{action} eq $Param::btn_Create){
   $tmp =$Param::RAdminPath."StateCreated.cgi";
   print <<END;
<FORM METHOD=POST ACTION=$tmp>
<CENTER>
<FONT COLOR=$Param::PromptColor><B>Creation of a new State.</B></FONT><BR><BR>
<TABLE>
<TR><TD ALIGN=right>State:</TD>
<TD><INPUT TYPE=text NAME=state MAXLENGHT=2 SIZE=2></TD></TR>
<TR><TD ALIGN=right>Name :</TD>
<TD><INPUT TYPE=text NAME=name MAXLENGTH=40 SIZE=40></TD></TR>
</TABLE>
<INPUT TYPE=submit VALUE=Commit><INPUT TYPE=reset VALUE=Reset>
</CENTER></FORM>
END
}

##########
##########
#Edit the selected state of the defect life cycle.
##########
##########
```

```
else {
   $lda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($lda,"select StateID, Name from State "
                 ."where (StateID = \'$fields{line}\')");
   @data=&ora_fetch($csr);
   foreach (@data){                                                    60
      tr/+/ /;
      s/\s*$//;
      s/%(..)/pack("c",hex($1))/ge;
   }
   ($state,$name)=@data;
   $tmp =$Param::RAdminPath."ReplyStt.cgi";
   print <<END;
<FORM METHOD=POST ACTION=$tmp>
<CENTER><TABLE>
<TR><TD ALIGN=right><B>State :</B></TD>                                70
<TD>$state<INPUT TYPE=hidden NAME=state VALUE=\"$state\"></TD></TR>
<TR><TD ALIGN=right><B>Name :</B></TD>
<TD><INPUT TYPE=text NAME=name VALUE=\"$name\" MAXLENGHT=40 SIZE=40></TD></TR>
</TABLE>
<INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_Cor\">
<INPUT TYPE=submit NAME=action VALUE=\"Delete\">
</FORM>
END
   &ora_close($csr);
   &ora_logoff($lda);                                                  80
}

print "</BODY></HTML>";
```

TblUnt.cgi

```
#!/usr/local/bin/perl
BEGIN {
        unshift (@INC, "../pub/lib/");
        unshift (@INC, "lib/");
        }

use Env;
use Oraperl;
use Param;
use AdminForm;                                                         10
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
        ($key,$content)=split(/=/,$item,3);
        $fields{$key}=$content;
        }
```

```
$fieℓds{action}=~tr/+/ /;
$fieℓds{action}=~s/%(..)/pack("c",hex($1))/ge;

print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text>
<CENTER>
END

##########
##########
#Display the form used to create a new Unit (or project).
##########
##########
if ($fieℓds{action} eq $Param::btn_Create){
   $tmp = $Param::RAdminPath."UnitCreated.cgi";
   print <<END;
<FORM METHOD=POST ACTION=$tmp>
<TABLE>
<TR><TD ALIGN=right>UnitID :</TD>
<TD><INPUT TYPE=text NAME=untid MAXLENGHT=30 SIZE=30></TD></TR>
<TR><TD ALIGN=right>Name :</TD>
<TD> <INPUT TYPE=text NAME=name MAXLENGTH=30 SIZE=30></TD></TR>
<TR><TD ALIGN=right>Release  :</TD>
<TD><INPUT TYPE=text NAME=rels MAXLENGTH=20 SIZE=20></TD></TR>
<TR><TD ALIGN=right>Responsible :</TD>
<TD><SELECT NAME=resp SIZE=1 >
END
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select GroupID  from GroupPers");
   while (($ℓine)=&ora_fetch($csr)){
      $ℓine=~tr/+/ /;
      $ℓine=~s/%(..)/pack("c",hex($1))/ge;
      print "<OPTION>$line";
   }
   print "</SELECT></TD></TR>";
   print "<TR><TD ALIGN=right>SubUnitOf :</TD><TD><SELECT NAME=sbnt SIZE=1 >";
   $csr=&ora_open($ℓda,"select UnitID  from Unit");
   print "<OPTION>none";
   while (($ℓine)=&ora_fetch($csr)) {
      $ℓine=~tr/+/ /;
      $ℓine=~s/%(..)/pack("c",hex($1))/ge;
      print "<OPTION>$line";
      }
   print "</SELECT></TD></TR>";
   &ora_close($csr);
   &ora_logoff($ℓda);
   print "</TABLE><INPUT TYPE=submit VALUE=Commit>";
   print "<INPUT TYPE=reset VALUE=Reset></FORM>";
   }
```

```
                                                                    70
##########
##########
#Edit the selected unit.
##########
##########
else {
   $ℓda=&ora_login('sid1','xgillman','xgillman');
   $csr=&ora_open($ℓda,"select UnitID, Name, Release, SubUnitOf, Responsible "
               ."from Unit where (UnitID =  \'$fields{line}\')");
   @data=&ora_fetch($csr);                                          80
   foreach (@data){
     tr/+/ /;
     s/%(..)/pack("c",hex($1))/ge;
     s/\s*$//;
     }
   ($untid,$name,$reℓ,$subunt,$resp)=@data;
   $tmp =$Param::RAdminPath."ReplyUnt.cgi";
   print "<FORM METHOD=POST ACTION=$tmp>";
   print "<TABLE><TR><TD ALIGN=right>ID :</TD><TD>$untid";
   print "<INPUT TYPE=hidden NAME=unitid VALUE=\"$untid\"></TD></TR>";   90
   print "<TR><TD ALIGN=right>Name :</TD>";
   print "<TD><INPUT TYPE=text NAME=name VALUE=\"$name\"></TD></TR>";
   print "<TR><TD ALIGN=right>Release :</TD>";
   print "<TD><INPUT TYPE=text NAME=rel VALUE=\"$rel\"></TD></TR><TR>";
   print "<TD ALIGN=right>Responsible :</TD><TD><SELECT NAME=resp SIZE=1 >";
   $csr=&ora_open($ℓda,"select GroupID from GroupPers");
   while (($option)=&ora_fetch($csr)){
      $option=~tr/+/ /;
      $option=~s/%(..)/pack("c",hex($1))/ge;
      $option=~s/\s*$//;                                            100
      if ($option eq $resp) {
         print "<OPTION SELECTED>$option";
         }
      else {
         print "<OPTION>$option";
         }
      }
   print "</SELECT></TD></TR><TR>";
   print "<TD ALIGN=right>SubUnitOf :</TD><TD><SELECT NAME=subunt SIZE=1 >";
   $csr=&ora_open($ℓda,"select UnitID from Unit where not(UnitID=\'$untid\')");   110
   print "<OPTION>none";
   while (($untid)=&ora_fetch($csr)) {
      $untid=~tr/+/ /;
      $untid=~s/%(..)/pack("c",hex($1))/ge;
      $untid=~s/\s*$//;
      if ($untid eq $subunt) {
         print "<OPTION SELECTED>$untid";
         }
      else {
         print "<OPTION>$untid";                                    120
```

```
        }
      }
   print "</SELECT></TD></TR>";
   print "</TABLE><INPUT TYPE=submit NAME=action VALUE=\"$Param::btn_Cor\">";
   print "<INPUT TYPE=submit NAME=action VALUE=\"Delete\">";
   print "</FORM>";
   }

print "</CENTER></BODY></HTML>";
```

UnitCreated.cgi

```
#!/usr/local/bin/perl
BEGIN {
      unshift (@INC, ". ./pub/lib/");
      unshift (@INC, "lib/");
      }

use Env;
use Oraperl;
use Param;
use AdminForm;
use DisplaySet;

read(STDIN,$temp,$ENV{'CONTENT_LENGTH'});
@pairs=split(/&/,$temp);
foreach $item(@pairs) {
  ($key,$content)=split(/=/,$item,3);
  $fields{$key}=$content;
  $fields{$key}=~s/%3C/&lt/g;
  }

##########
#Create a new record in the database.
##########
$lda=&ora_login('sid1','xgillman','xgillman');
if ($fields{sbnt}= 'none'){
   $csr=&ora_open($lda,"insert into Unit (UnitID, Name, Release, Responsible)"
               ." values (\'$fields{untid}\',\'$fields{name}\',"
               ."\'$fields{rels}\',\'$fields{resp}\')");
   }
else {
   $csr=&ora_open($lda,"insert into Unit (UnitID, Name, Release, Responsible,"
               ."SubUnitOf ) values (\'$fields{untid}\',\'$fields{name}\',"
               ."\'$fields{rels}\',\'$fields{resp}\',\'$fields{sbnt}\')");
   }
print <<END;
Content-type: text/html\n\n
<HTML><BODY $Param::bgcolor $Param::text><CENTER>
<FONT COLOR=$Param::PromptColor>
```

```
<B>Unit <FONT COLOR=#000000>$fieℓds{untid}</FONT> created.</B></FONT><BR>
END

##########
#Display the unit table for administration.
##########
$csr=&ora_open($ℓda,'select UnitID, Name, Release, SubUnitOf, Responsible"
              ."from Unit');
@col=(UnitID, Name, Release, SubUnitOf, Responsible);
&DisplaySet::display($csr,\@col,$Param::RAdminPath."TblUnt.cgi",
                    $Param::btn_Create);
&ora_close($csr);
&ora_logoff($ℓda);

&AdminForm::display();
print "</CENTER></BODY></HTML>";
```